Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Proyecto Final de Carrera
Ingeniería Informática
Curso 2011-2012

Análisis y desarrollo de una aplicación web para la publicación y venta de productos.

Memoria

Daniel Fuertes Pueyo

Junio de 2012

Director: Antonio Carro Mariño

Responsable de estrategia e innovación
Sdweb Soluciones Digitales

Ponente: Santiago Velilla Marco

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Índice

Índice de tablas

Índice de ilustraciones

1 INTRODUCCIÓN

El objetivo de esta memoria es explicar del modo mas simple y claro posible el objeto del proyecto y los aspectos mas relevantes de su diseño, implementación y prueba.

Este capítulo pretende introducir al lector en el dominio del sistema desarrollado. Se presentará el contexto en el que ha sido desarrollado, además de enumerar los objetivos del proyecto y las tareas realizadas dentro del mismo.

1.1 Contexto

Este proyecto ha sido desarrollado en la empresa Sdweb Soluciones Digitales. Sdweb es una pequeña empresa en crecimiento situada en la ciudad de Santiago de Compostela. Trabaja sobre todo con las instituciones (ayuntamientos, universidades,…) de la zona de Galicia, aunque también tiene proyectos en Cataluña. Se dedica a tres campos principalmente:

- Comunicación digital: proyectos adaptados a necesidades concretas, optimización de entornos y desarrollo de estrategias de comunicación con el fin de aumentar su visibilidad.
- E-learning: social learning y potenciación de la comunicación y colaboración en el seno de las organizaciones.
- Gestión documental: Desarrollo e implementación de sistemas de gestión documental y bases de datos para garantizar la calidad en la gestión de la información.

Las posibilidades de acceso a la información que ofrecen las tecnologías actuales (internet, telefonía móvil…) ha propiciado que un gran número de empresas, particulares e instituciones, intenten ofrecer sus productos y servicios a todos sus potenciales consumidores de forma atractiva y eficaz, utilizando estas tecnologías. Esto ha hecho que hayan aparecido en el mercado diferentes "sistemas de ofertas", como Groupon, Oooferton, Ofertix o Menus.es, dedicados a promocionar y mejorar las ventas de las empresas/negocios adheridos a dichos sistemas. No obstante, para muchos pequeños comercios, el coste que representa adherirse a uno de estos sistemas puede resultar demasiado elevado (bien sea porcentaje sobre cada venta, precio por publicación, etc.).

Por todos estos motivos surgió la idea de este proyecto, como solución a un problema real, en el que un Ayuntamiento pidió a Sdweb una **central de ofertas** que permitiera a sus comerciantes ofertar sus productos y servicios de una manera económica, y que además contara con un apartado para que las instituciones del Ayuntamiento puedan publicar eventos de interés público, como inauguraciones, cortes en carreteras, etc. Así, los ciudadanos de dicha localidad y su entorno tendrán la posibilidad, bien desde su casa o un teléfono móvil, de realizar sus compras y estar informados de los eventos de interés programados.

1.2 Objetivos y alcance del proyecto

El objetivo de este proyecto es diseñar y desarrollar un sitio web que permita la compra-venta de productos y servicios, así como la visualización de eventos y noticias de interés de las empresas e instituciones que pertenezcan al entorno del Ayuntamiento. Para completar la aplicación, se dispondrá de un apartado, con interfaz separada, que implemente un sistema de ofertas que permita que cualquier empresa particular o institución, a nivel nacional y una vez dado de alta en el sistema, pueda ofertar sus productos con un coste reducido, servicios y eventos de modo que cualquier potencial consumidor pueda informarse o comprar, desde internet o desde un teléfono móvil. Es en este apartado donde las empresas e instituciones del entorno del ayuntamiento podrán gestionar sus ofertas y eventos. Por otra parte, se intentará diseñar un producto lo más genérico posible, de modo que pueda ser utilizado, con un coste de adaptación reducido, para dar soporte a otra empresa/entidad.

El sitio web estará alojado en un servidor de Sdweb y sera administrado por ésta.

1.3 ¿A quien va dirigido?

La central de ofertas no está acotada a ningún sector en concreto y su ámbito (idiomas) de implantación sería el nacional (español, gallego) aunque el sistema desarrollado estará implementado de tal manera que se sea fácilmente traducible a cualquier otro idioma.

- Empresa pequeña – mediana que no cuenta con un portal web y sólo desea publicar información.
- Empresa pequeña – mediana que tiene un portal web y desea compaginarlo con una aplicación móvil.
- Empresa de nueva creación que desee contar con un sistema que le permita fácilmente incorporar ofertas y publicarlas en un canal de ofertas común
- Asociaciones de todo tipo
- Centros comerciales, mercados, centros de negocio…
- Administraciones locales para potenciar la industria turística, gastronómica, termal, ofrecer información sobre lo que sucede en un lugar concreto
- Promotores inmobiliarios
- Empresa pequeña – media – grande que necesite realizar una campaña de marketing a través de internet para dar a conocer sus productos o servicios en detalle y articular fórmulas para contactar con los clientes.

1.4 Ciclo de vida

El primer paso consistirá, además de la planificación del desarrollo del proyecto, en el análisis de requisitos y la elaboración de las especificaciones de detalle, en base a las necesidades del cliente y de los usuarios. Esto incluirá una comparación de los sistemas de ofertas más relevantes que hay en el mercado para extraer ideas y elaborar un proyecto competitivo. También se realizará un análisis de las diferentes tecnologías existentes que podrían utilizarse, atendiendo a criterios de coste, tiempo y facilidad de aprendizaje.

Una vez establecidos los requisitos del proyecto se iniciará el diseño del sistema, prestando especial atención al diseño de las interfaces de usuario para navegador y para móvil, y de la base de datos y estructuras de información necesarias.

Tras la implementación, utilizando las herramientas y lenguajes seleccionados, se comprobará el correcto funcionamiento del sistema y el cumplimiento de los requerimientos previamente establecidos.

El ciclo de vida seleccionado para el desarrollo del sistema ha sido el ciclo de vida en cascada mejorado.

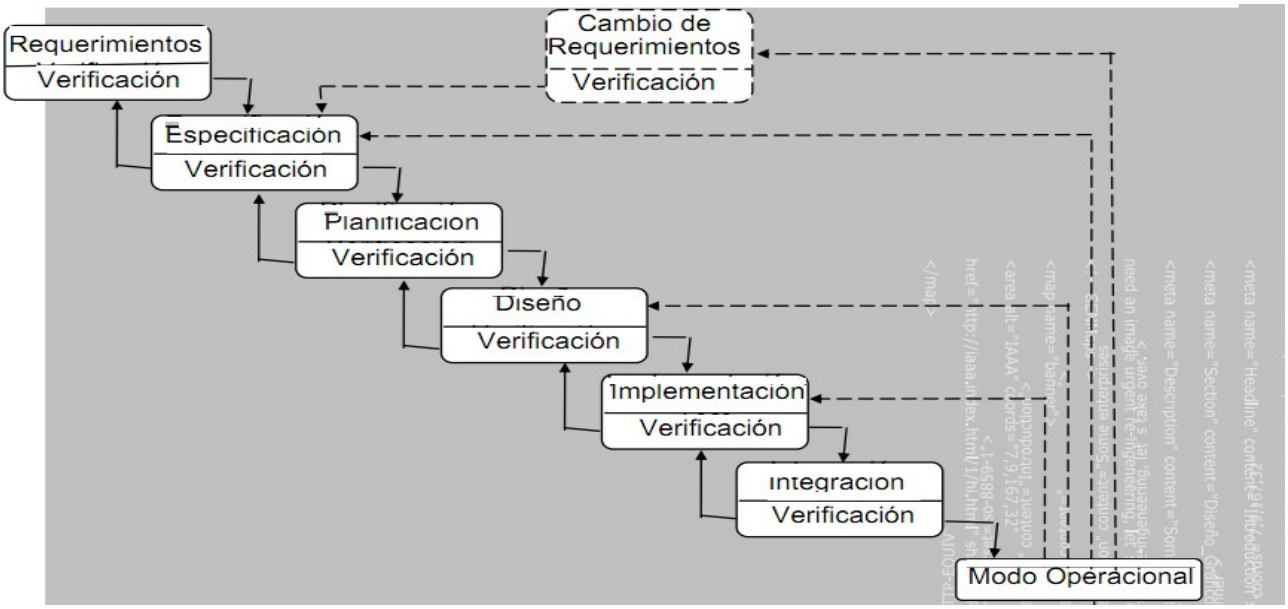


Ilustración 1: ciclo de vida en cascada

Es un ciclo de vida que admite iteraciones, contrariamente a la creencia de que es un ciclo de vida secuencial o lineal. Después de cada etapa se realiza una o varias revisiones para comprobar si se puede pasar a la siguiente. Aun asi, es un modelo rígido, poco flexible y con muchas restricciones.

Una de sus ventajas, ademas de su planificación sencilla, es la de proporcionar un producto final con un elevado grado de calidad, sin necesidad de un personal altamente cualificado. Se puede considerar como inconvenientes la necesidad de contar con todos los requerimientos (o la mayoría) al comienzo del proyecto y que, si se han cometido errores y no se detectan en la etapa siguiente, es costoso y complejo volver atrás para realizar la corrección del error. Además, los resultados no los veremos hasta que no estemos en las etapas finales del ciclo, por lo que cualquier error detectado genera un retraso y aumenta el costo del desarrollo de forma proporcional al tiempo que empleado en la corrección de dichos errores.

Es un ciclo de vida adecuado para los proyectos en los que se dispone de todos los requerimientos al comienzo, para el desarrollo de un producto con funcionalidades conocidas o para proyectos que, aun siendo muy complejos, se entienden perfectamente desde el principio.

Se ha elegido este ciclo de vida porque desde el inicio se dispone de todos los requerimientos y se conocen sus funcionalidades. Además, el Ayuntamiento no tiene prisa por ver el resultado, ya que en el momento del desarrollo del proyecto no tenía presupuesto para pagarlo. El motivo por el que se ha implementado este sistema es que Sdweb ha visto una oportunidad de negocio en este proyecto, ya que el sistema de ofertas nacional será administrado por Sdweb y el sistema de ofertas especifico de un ayuntamiento, al ser genérico, podría ser vendido a otro ayuntamiento.

1.5 Planificación

En este apartado se va a explicar la planificación de tareas que se ha seguido durante el proyecto. Para ello, se expone un diagrama de Grantt que ilustra la realización de las diferentes fases que componen el proyecto.

Para comenzar se hizo una estimación del coste en tiempo de la fase de análisis y diseño. Una vez realizada esta fase, se pudo completar el diagrama de Grantt con las siguientes fases del proyecto.

Nombre	Duración	Inicio	Terminado
Análisis preliminar y prupuesta	7 days?	26/09/11 7:00	4/10/11 17:00
Requisitos	1 day?	5/10/11 7:00	5/10/11 17:00
⊞Análisis y diseño	11 days	**6/10/11 7:00**	**20/10/11 17:00**
⊞Implementacion	115 days?	**21/10/11 7:00**	**29/03/12 17:00**
⊞Plan de pruebas	6 days	**30/03/12 7:00**	**6/04/12 17:00**
⊞Documentación	10 days	**7/04/12 7:00**	**20/04/12 17:00**

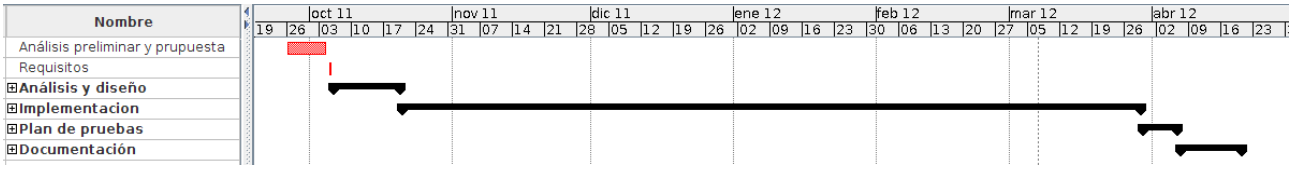


Ilustración 2: Planificación

1.6 Organización de este documento.

El presente documento está dividido en dos partes: Memoria, donde se explica el desarrollo del proyecto, y Anexos, donde se amplía la información de ciertos puntos relevantes.

En el siguiente capítulo se realiza una investigación preliminar sobre los diferentes sistemas de ofertas que existen en el mercado y sobre las tecnológias que se van a utilizar. Además se realiza

un estudio sobre el tipo de consumidor al que le podría interesar el producto desarrollado.

Posteriormente se muestran los requisitos funcionales y no funcionales del sistema extraídos de la investigación preliminar.

Una vez presentados los requisitos, se realiza el análisis y el diseño del sistema. En este capitulo se pretende definir de forma clara y precisa lo que desea el usuario y la forma en la cual se va a presentar la solución que se está buscando y se define una subdivisión en aplicaciones del sistema.

Tras el diseño, se presentan los aspectos de la implementación más importantes y que como desarrollador han supuesto un mayor desafío o han resultado más llamativos o novedosos.

Después de haber explicado la implementación, se describen las pruebas realizadas para comprobar el cumplimiento de los requerimientos.

En el ultimo capítulo se exponen las conclusiones y las líneas futuras de desarrollo.

2 INVESTIGACIÓN PRELIMINAR

2.1 Comparación de los sistemas de ofertas más relevantes

El primer paso antes de abordar el diseño de un sistema suele consistir en la evaluación critica y estudio comparativo de los sistemas más relevantes similares al que se va a desarrollar.

En este apartado se va a realizar un estudio de los aspectos generales de los sistemas de ofertas mas relevantes para extraer ideas, no sólo acerca de aspectos funcionales, estéticos y de interfaz, sino también de herramientas, con objeto de elaborar un proyecto competitivo.

Las webs que ofrecen productos y servicios a precios rebajados o con descuento han inundado el mercado actual, quizás debido a la crisis que nos acompaña desde hace tiempo. Son una buena oportunidad para comprar dichos productos y servicios con descuento de manera cómoda y sencilla desde nuestros hogares, pero adolecen todavía de algunos inconvenientes que deberemos tener en cuenta.

Existen dos tipos claramente diferenciados, las webs de compra directa y las de compra colectiva. Las webs de compra directa, son clubs de venta privada online que ofrecen artículos rebajados o con descuento. Su ámbito es local y se encargan de enviar por correo al domicilio del comprador dichas ofertas. Algunos también venden servicios (como estancias en hoteles o viajes). Las páginas mas conocidas son Privalia, ofertix y BuyVip.

Por otro lado, las webs de compra colectiva son intermediarios que venden cupones canjeables por servicios en otras empresas y sus ofertas cambian en función del número de personas que se apunte a ellas. Normalmente requieren de un número mínimo de personas “apuntadas” a las ofertas para que éstas se materialicen. Las páginas mas conocidas son Groupon, LetsBonus y Planeo.

Como resumen de las características básicas de los sistemas de ofertas que hay en el mercado, en la tabla que se presenta a continuación se muestran dichas características, junto con las que dispondrá el sistema de central de ofertas a desarrollar.

Características	**Central de ofertas**	**compras colectivas**	**Compras directas**
Empresa			
La empresa puede publicar sus ofertas a través del portal.	Sí	Sí	Sí
Las ventas de ofertas se realizan a través del portal.	Sí	Sí	Sí

Características	Central de ofertas	compras colectivas	Compras directas
Importes de ventas recibidos por el portal van directamente a la cuenta de la Empresa.	Sí	No	No
Gestión de ofertas de forma autónoma por la empresa.	Sí	No	No
Comisión a pagar por venta	10%	50%	No
Precio por publicación	No	No	Sí
Ver ventas realizadas	Sí	Sí	Sí
Ver compras realizadas	Sí	No	No
Ver ofertas publicadas	Sí	Sí	Sí
Entrega a domicilio	No	No	Algunas
Institución			
La institución puede publicar sus eventos a través del portal	Sí	No	No
Gestión de eventos de forma autónoma por la empresa.	Sí	No	No
Ver eventos publicados	Sí	No	No
Modificar datos	Sí	No	No
Usuario			
Ofertas del día	Sí	Sí	Sí
¿Que hay a mi alrededor?	Sí	Sí	Sí
Guía de mi ciudad	Sí	No	No
Buscar reservas	Sí	No	No
Añadir oferta a favoritos	Sí	No	No
Guardar búsqueda	Sí	No	No
Ver compras realizadas	Sí	Sí	Sí
Modificar datos	Sí	Sí	Sí

Tabla 1: central de ofertas vs sistemas de compra directa

Mientras que las web de compra directa están enfocadas a la publicidad de las empresas para darlas a conocer, obteniendo un alto porcentaje por cada venta, la central de ofertas está más enfocada a vender productos y servicios a partir de un método de publicación de ofertas más barato y menos controlado que los otros dos. Además, el sistema desarrollado ofrece al consumidor más opciones a la hora de la buscar ofertas, ya que cuenta con un buscador de ofertas, mientras que en los otros dos sólo puedes ver las ofertas del día de una localidad.

El sistema desarrollado también cuenta con un apartado para instituciones donde pueden gestionar sus eventos de manera gratuita.

De este estudio se extrajeron los requisitos del proyecto.

2.2 Análisis de tecnologías

En este apartado se va a realizar un análisis de las diferentes tecnologías existentes que podrían utilizarse, atendiendo a criterios de coste, tiempo y facilidad de aprendizaje.

Las tecnologías utilizas para el desarrollo de la central de ofertas han sido:

- Sistema operativo: Ubuntu 11.4
- Programación
    - Entorno de desarrollo: Eclipse Indigo
    - Base de datos: PostgreSQL 8.4
    - Lenguaje: Java 6
    - Framework RIA: Vaadin 6.7.1
    - Framework mapeo objeto-relación: ORMLite 4.29
    - Tomcat 7.0
- Herramientas de producción documental
    - LibreOffice 3.3.4
    - Balsamiq Mockups 2.1.14
    - ArgoUML 0.32.2

La primera tecnología que había que elegir para la implementación del sistema era un entorno de desarrollo que ayudara a trabajar con código fuente y que incluyera un debug (para revisar errores en el código), un analizador de sintaxis y un precompilador (para detectar errores previos a la compilación). El entorno de desarrollo elegido fue la versión Indigo de **Eclipse** ya que cuenta con un componente para trabajar con Vaadin.

2.2.1 Framework RIA: Vaadin

Un framework RIA (*Rich Internet Applications*) es un framework diseñado para apoyar el desarrollo de sitios webs dinámicos, aplicaciones web y servicios web. Este tipo de frameworks intenta aliviar el exceso de carga asociado con actividades comunes usadas en desarrollos web.

Vaadin es un framework para la plataforma java, de código abierto, para la creación de aplicaciones web. A diferencia de las soluciones basadas en plugins del navegador, Vaadin contempla una arquitectura en la que la mayor parte de la lógica recae en el servidor. Se utiliza tecnología Ajax para asegurar una experiencia de usuario interactiva. Además, utiliza Google Web Toolkit como base para el desarrollo del lado del cliente.

Vaadin incluye una gran colección de componentes para interfaces de usuario. A la hora de generar la vista de la aplicación, se utilizan simples botones, tablas y plantillas que se comunican entre sí y con la lógica de la aplicación mediante eventos, listeners y enlaces de datos. Esta arquitectura basada en componentes facilita la creación de aplicaciones modulares y fácilmente refactorizables. Hay complementos adicionales para entornos de desarrollo (*IDE - Integrated Development Environment*) que facilitan el diseño de manera visual y la realización rápida de prototipos.

Las principales **ventajas** que ofrece son:

- La curva de aprendizaje no es demasiado elevada ya que se utiliza exclusivamente código Java.

- Vaadin tiene un interesante sistema de add-ons (componentes para extender las funcionalidades de Vaadin). Uno de estos add-ons, Vaadin touchKit, ofrece la posibilidad de crear una aplicación para dispositivos móviles (android, iOs,...) a partir de una aplicación web.
- Vaadin permite integrar nuestra aplicación directamente en código html.
- Implementa el patrón MVC (Modelo-Vista-Controlador).

Desde un principio se tuvo claro que se quería hacer una aplicación web, ya que el sistema tenía mucho de gestión (gestión de usuarios, empresas, ofertas, eventos,...) y ademas se quería una aplicación que se pudiera usar tanto desde un computador como desde un dispositivo móvil (smartPhone, tablet,...). También se quería una aplicación para integrar en el sitio web del ayuntamiento. En estos términos, el framework Vaadin parece la tecnología ideal para la implementación del sistema frente a otros frameworks más utilizados como son Struts, Java Server Faces o Spring MVC.

2.2.2 Framework ORM: ORMLite vs Hibernate

Un framework ORM (Object Relational Mapping**)** es un framework diseñado para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. Los frameworks a estudiar son ORMLite debido a que esta especializado en aplicaciones móviles e Hibernate debido a que es la más utilizada.

ORMLite proporciona algunas funcionalidades de peso ligero para la persistencia de objetos Java a bases de datos SQL, evitando la complejidad y la sobrecarga de otros paquetes ORM estándar.

ORMLite ofrece las siguientes características:

- Configuración de sus clases por simple adición de anotaciones Java.
- Potente acceso a base de datos a través de objetos DAO.
- Generador de consultas flexibles para construir fácilmente consultas simples y complejas.
- Soporta MySQL, PostgreSQL y Microsoft SQL.
- Soporte básico para las transacciones de base de datos.

Hibernate es una herramienta ORM completa que ha conseguido en un tiempo récord una excelente reputación en la comunidad de desarrollo, posicionándose claramente como el producto OpenSource líder en este campo, gracias a sus prestaciones, buena documentación y estabilidad. Es valorado por muchos incluso como solución superior a productos comerciales dentro de su enfoque, siendo una muestra clara de su reputación y soporte la reciente integración dentro del grupo JBOSS que seguramente generará iniciativas muy interesantes para el uso de Hibernate dentro de este servidor de aplicaciones.

El mayor inconveniente es que tiene es que es demasiado pesada (casi 3MB) para el desarrollo de una aplicación para móviles. Esta razón ha sido suficiente para descartarla y elegir ORMLite para el desarrollo del sistema, ya que uno de los requisitos del proyecto es que pueda accederse al sistemas desde un dispositivo móvil.

2.2.3 Base de datos: PostgreSQL vs MySQL

En este apartado se van a comparar dos de las bases de datos mas utilizadas, PostgreSQL, porque es de código abierto y es la que se utilizaba en Sdweb y MySQL porque es la mas utilizada en el mundo del software libre.

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (ORDBMS) basado en el proyecto POSTGRES, de la universidad de Berkeley, siendo una derivación libre (OpenSource) de este proyecto. Utiliza el lenguaje SQL92/SQL99.

Fue el pionero en muchos de los conceptos existentes en el sistema objeto-relacional actual, incluido más tarde en otros sistemas de gestión comerciales. Incluye características de la orientación a objetos, como puede ser la herencia, además de tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional.

MySQL es un sistema de gestión de bases de datos relacional, licenciado bajo la GPL de la GNU. Su diseño multihilo le permite soportar una gran carga de forma muy eficiente. MySQL fue creada por la empresa sueca MySQL AB, que mantiene el copyright del código fuente del servidor SQL, así como también de la marca.

Este gestor de bases de datos es, probablemente, el gestor más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. Esta gran aceptación es debida, en parte, a que existen infinidad de bibliotecas y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración.

Las **características positivas** que posee **PostgreSQL** son:

- Posee una gran escalabilidad. Es capaz de ajustarse al número de CPUs y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta (en algunos casos ha llegado a soportar el triple de carga de lo que soporta MySQL).
- Esta diseñado para ambientes de alto volumen de datos. PostgreSQL usa una estrategia de almacenamiento de filas llamada MVCC para conseguir una respuesta más eficiente en ambientes de grandes volúmenes de datos. Los principales proveedores de sistemas de bases de datos comerciales usan también esta tecnología, por las mismas razones.

Por contra, los **mayores inconvenientes** que se pueden encontrar a este gestor son:

- Consume gran cantidad de recursos.
- Es de 2 a 3 veces más lento que MySQL.

Las **características positivas** que posee **MySQL** son:

- Sin lugar a duda, lo mejor de MySQL es su velocidad a la hora de realizar las operaciones, lo que le hace uno de los gestores que ofrecen mayor rendimiento.
- Su bajo consumo de recursos lo hace apto para ser ejecutado en una máquina con escasos recursos sin ningún problema.
- Las utilidades de administración de este gestor son envidiables para muchos de los gestores comerciales existentes, debido a su gran facilidad de configuración e instalación.
- Tiene una probabilidad muy reducida de corromper los datos, incluso en los casos en los que los errores no se produzcan en el propio gestor, sino en el sistema en el que está.

Por contra, los **mayores inconvenientes** que se pueden encontrar a este gestor son:

- Carece de soporte para transacciones, rollback's y subconsultas.
- No es viable para su uso con grandes bases de datos, a las que se acceda continuamente, ya que no implementa una buena escalabilidad.

Como conclusión a la comparación entre MySQL y PostgresSQL, podemos decir que MySQL es mejor para sitios web con contenido dinámico, discusiones, noticias, etc. En general, MySQL se considera más eficiente para sistemas en los que la velocidad y el número de accesos concurrentes sea algo primordial, y la seguridad no sea muy importante.

En cambio, para sistemas más serios en las que la consistencia de la base de datos sea fundamental y haya un gran volumen de información, PostgreSQL es una mejor opción pese a su

mayor lentitud.

La idea era que la central de ofertas contara con un gran volumen de usuarios y que estos accedieran de manera concurrente al sistema por lo que se creyó que la decisión mas acertada era usar PostgreSQL para la implementación de la base de datos.

3 REQUISITOS

3.1 Requisitos funcionales

La central de ofertas implementará un sistema de ofertas que permita que cualquier empresa particular o institución, a nivel nacional, una vez dado de alta en el sistema, y con un coste reducido, pueda ofertar sus productos, servicios y eventos para que el consumidor pueda informarse o comprar, desde internet o desde un teléfono móvil. La central de ofertas también contará con un tipo de usuario que sera el encargado de administrar el sistema.

RF-1: Habrá 4 tipos de usuarios: usuario, empresa, institución y administrador. Cada uno tendrá su apartado con interfaz gráfica diferente. Al acceder a la aplicación se mostrará el interfaz correspondiente con el tipo de usuario.

De cara a la **empresa**

RF-2: Cualquier tipo de empresa puede publicar sus ofertas. Para publicar las ofertas necesita estar registrada en el sistema y se necesita verificar la existencia real de esa empresa ya que hay pagos en su cuenta.

RF-3: Para registrarse debe dar un nombre de empresa, una clave, una dirección de correo electrónico, un teléfono para tener sus datos de contacto y poder verificar la cuenta.

RF-4: Para identificarse en el sistema usará su dirección de correo electrónico y clave.

Cada empresa registrada puede gestionar su central de reservas. El objetivo es que cada una de las empresas registradas tenga un panel de control a través del cual pueda:

RF-5: Incorporar y modificar los datos de su empresa. Una empresa después de registrarse debe rellenar unos campos mientras espera a ser validada por el administrador. Los datos son NIF o DNI (* Los autónomos utilizan DNI), nombre y apellidos del administrador o gerente,país (*), provincia (*), comarca (*), código postal (*), número de cuenta (* para recibir los cobros de las reservas y para poder cobrar mensualmente el porcentaje generado). Estos datos son los que se utilizarán para la facturación (tanto de las facturas que ellos emitan como las facturas que reciban).

RF-6: Incorporar las sedes de la empresa. Una empresa siempre tiene que indicar como mínimo (1:N) las sedes que la conforman. Cada sede tendría los siguientes campos: provincia (*), comarca (*), ayuntamiento (*), código Postal (*), calle (*), número (*), piso.

RF-7: Publicar ofertas. Una empresa puede publicar las ofertas que desee teniendo en cuenta los siguientes campos: título, descripción, imagen, precio, iva aplicado, sector, stock inicial, sedes asociadas a la oferta (1 sede siempre mínimo), fecha en que se publica, fecha en que finaliza, tipo de oferta, privacidad de la oferta

RF-8: Privacidad de la oferta; *Oferta pública* / *oferta privada* (la empresa puede incluir diferentes ofertas y seleccionar si esta oferta es pública y aparece en el canal o es una oferta privada para tener un control de su gestión interna).

RF-9: Hay 2 tipos de ofertas diferentes:

- *Oferta básica*: sólo se ofrece información.
- *Oferta con reserva:* se ejecuta la compra a través de una pasarela de pago seguro. El importe de la reserva se ingresa en la cuenta de la empresa.

Nota: * significa campo obligatorio.

RF-10: Listado secuencial de las reservas publicadas: en este caso es básico que se muestren las reservas que la empresa ha indicado inicialmente y las que quedan. Si una empresa realiza una compra, el número de reservas decrece.
RF-11: Calendario con el total de las reservas incorporadas (pública y privada)
RF-12: modificar/eliminar ofertas
RF-13: Histórico de facturas generadas. El histórico de facturas generadas coincide con los tickets que se envían al cliente final. Es decir el ticket es una factura con los datos de su compra y los datos de la empresa.

De cara a las **instituciones**

RF-14: Cualquier tipo de institución puede publicar sus eventos. Para publicar las ofertas necesita estar registrada en la misma y se necesita verificar la existencia real de esa institución.
RF-15: Para registrarse debe dar un nombre de institución, una clave, una dirección de correo electrónico, un teléfono para tener sus datos de contacto y poder verificar la cuenta.
RF-16: Para identificarse en el sistema usará su dirección de correo electrónico y clave.

Cada institución registrada puede gestionar su central de eventos. El objetivo es que cada una de las instituciones registradas tenga un panel de control a través del cual pueda:

RF-17: Incorporar y modificar sus datos. Una institución después de registrarse debe rellenar unos campos mientras espera a ser validada por el administrador. Los datos que debe incorporar son: Nombre y apellidos del administrador o gerente, país (*), provincia (*), comarca (*), código postal (*)
RF-18: Publicar eventos. Una institución puede publicar los eventos que desee teniendo en cuenta los siguientes campos: Título, descripción, imagen, fecha, lugar, duración.
RF-19: Listado secuencial de los eventos publicados: en este caso es básico que se muestren los eventos que la institución ha indicado inicialmente y los que quedan.
RF-20: Visualización de los eventos en forma de calendario.
RF-21: modificar/eliminar eventos

De cara al **usuario***:*

RF-22: El usuario podrá acceder desde un navegador web o desde un dispositivo móvil para visualizar las ofertas y eventos a través de diferentes conceptos.
RF-23: No es preciso que el usuario se registre para ver las ofertas, sino sólo en el caso de que desee reservar alguna de ellas. El sistema de registro debe ser muy simple (Nombre, dirección de correo electrónico y clave).
RF-24: Si el usuario intenta realizar una compra sin haberse identificado, se le pedirá al usuario que se identifique si tiene cuenta o que se registre si no la tiene.
RF-25: Un usuario se identificará en el sistema mediante su dirección de correo electrónico y su clave.
RF-26: Si el usuario entra desde el móvil, se detectara la provincia en la que se encuentra y la elegirá por defecto para las diferentes búsquedas.

El usuario podrá realizar búsquedas de ofertas por los siguientes criterios:

RF-27: Buscar reservas por diferentes criterios (Provincia, ciudad, sector, palabras clave, descripción y empresa)
RF-28: Ofertas del día
RF-29: Ofertas de tu provincia.
RF-30: Guía de mi provincia (eventos de mi provincia)
RF-31: Una vez que se ha realizado una búsqueda de eventos, el usuario podrá acceder a un evento. El detalle de un evento se compone de los siguientes campos: título, descripción, fecha,

imagen, lugar y duración.

RF-32: El usuario podrá ver el detalle de la institución que publica el evento.

RF-33: Una vez que se ha realizado una búsqueda de ofertas, el usuario podrá acceder a una oferta. El detalle de una oferta se compone de los siguientes campos: título, descripción, etiquetas, imagen, precio, tiempo hasta que caduca la oferta, stock, sedes en las que está publicada y el tipo de oferta y comentarios.

RF-34: En el caso de que sea con reserva, se mostrara al usuario un botón "Compra".

RF-35: El usuario podrá ejecutar la compra indicando el numero de ofertas a comprar y los datos de su tarjeta de crédito: nombre del titular, número de tarjeta, fecha de caducidad y código. La operación de compra se realizara a través de una pasarela de pago seguro ofrecida por un banco ingresando el dinero directamente a la cuenta del vendedor.

RF-36: En el momento que el usuario realiza el pago, él y el vendedor, reciben en su correo electrónico (dato obligatorio) un ticket de recepción del pedido modelo factura que recoge: datos de la empresa, datos del cliente y los datos de la oferta. Ese ticket habrá que entregarlo a la hora de recoger la compra.

RF-37: De forma paralela, la empresa y el comprador reciben en su panel de gestión el aviso de la reserva efectuada

Además el usuario tiene las siguientes funcionalidades con respecto a las ofertas:

RF-38: El usuario podrá ver el detalle de la empresa que la publica.

RF-39: El usuario podrá añadir una oferta a sus favoritos

RF-40: El usuario podrá eliminar una oferta de sus favoritos

RF-41: El usuario podrá guardar búsquedas favoritas.

RF-42: Una búsqueda se compondrá de los siguientes criterios: Provincia, ciudad, sector, palabras clave, descripción y empresa.

RF-43: El usuario podrá publicar el enlace a las redes sociales (Facebook, Twitter)

RF-44: El usuario podrá ver y añadir comentarios asociados a la oferta

RF-45: El usuario podrá modificar/eliminar los comentarios que haya hecho a una oferta

RF-46: El usuario podrá hacer una búsqueda por vinculación semántica de palabras palabras clave

RF-47: El usuario podrá ver mapa con la localización de las sedes que contienen la oferta.

Los usuarios registrados tendrán un panel de control a través del cual podrán:

RF-48: Modificar sus datos de usuario (importante si quiere gestionar facturas). Los datos a completar son: país, provincia, comarca, ciudad, código Postal, teléfono, fecha de nacimiento

RF-49: Ver el histórico de compras

RF-50: Ver sus ofertas favoritas

RF-51: Ver sus búsquedas favoritas

RF-52: Ver empresas favoritas (empresas con las que ha hecho más operaciones de compra)

De cara al **administrador**:

RF-53: No se podrá realizar un registro como tipo de usuario *administrador*, por lo que las credenciales del administrador se introducen de forma manual en la base de datos.

RF-54: Una vez se haya identificado podrá cambiar la contraseña

El administrador tiene labores de gestión de empresa, instituciones, usuarios, ofertas y eventos. Dispondrá de un panel en el que podrá:

RF-55: Aceptar/denegar peticiones de nuevas empresas

RF-56: Una vez aceptada una petición se le enviará un correo electrónico a la empresa

avisándole.
RF-57: Buscar/eliminar empresas que hagan uso indebido de la aplicación.
RF-58: Listar ofertas de una empresa
RF-59: Listar ventas de una empresa
RF-60: Calcular un porcentaje sobre los ingresos que ha recibido una empresa en un determinado mes.
RF-61: Aceptar/denegar peticiones de nuevas instituciones
RF-62: Una vez aceptada una petición se le enviara un correo electrónico a la institución avisándole.
RF-63: Buscar/eliminar instituciones que hagan uso indebido de la aplicación.
RF-64: Listar eventos de una institución
RF-65: Buscar/eliminar usuarios que hagan uso indebido de de la aplicación
RF-66: Listar compras de un usuario
RF-67: Buscar/eliminar reservas
RF-68: Buscar/eliminar eventos

Requisitos sobre **validación** de datos.

RF-69: No podrán registrarse dos usuarios de cualquier tipo con la misma dirección de correo electrónico (aunque sean distinto tipo de usuario).
RF-70: No podrán registrarse dos empresas con el mismo nombre.
RF-71: No podrán registrarse dos instituciones con el mismo nombre.
RF-72: Ningún tipo de usuario podrá registrarse con el mismo nombre que el administrador.
RF-73: Todos los campos de registro deberán ser obligatorios.
RF-74: Una dirección de correo electrónico deberá tener el siguiente formato: buzon**@**subdominio**.** ... **.**subdominio2**.**subdominio1**.**dominio-de-mas-alto-nivel
RF-75: EL teléfono, el código postal, el numero de cuenta, el portal, el piso, el precio, el iva y el stock inicial deberán ser un número.

Además la central de ofertas dispondrá de un apartado, con interfaz separada, para integrarse en un sitio web que permita la compra-venta de productos y servicios y la visualización de eventos y noticias de interés de las empresas e instituciones que pertenezcan al entorno de un Ayuntamiento específico.

Este apartado podrá utilizarse sin identificarse en el sistema, solo en caso de querer comprar alguna reserva habrá que realizar el acceso.

En este apartado un usuario podrá:

RF-76: buscar reservas del día de una zona
RF-77: buscar eventos de una zona
RF-78: buscar reservas por diferentes criterios(empresa, descripción, sector, etiquetas)
RF-79: Buscar empresas de la zona
RF-80: buscar reservas del día de una empresa de la zona
RF-81: buscar reservas de una empresa de una zona por diferentes criterios(empresa, descripción, sector, etiquetas)
y los requisitos RF-25, RF-33, RF-34, RF-35, RF-36, RF-37, RF-38, RF-39, RF-40.

3.2 Requisitos no funcionales

RNF-1: Garantizar la confiabilidad, la seguridad y el desempeño del sistema informático a los diferentes usuarios a nivel nacional. En este sentido la información almacenada podrá ser consultada y actualizada permanente y simultáneamente, sin que se afecte el tiempo de respuesta.
RNF-2: El sistema debe tener capacidad para dar respuesta al acceso simultaneo de todos los usuarios con tiempo de respuesta aceptable y uniforme en períodos de alta, media y baja

demanda de uso del sistema.

RNF-3: El sistema debe estar implementado de tal manera que la incorporación de nuevas funcionalidades y requerimientos pueda realizarse de modo que que el código existente se vea afectado de forma mínima.

RNF-4: El sistema debe ser de fácil uso por parte de los usuarios.

RNF-5: El sistema debe presentar mensajes de error que permitan al usuario identificar el tipo de error y comunicarse con el administrador del sistema.

RNF-6: El sistema debe contar con facilidades para la identificación de la localización de los errores durante la etapa de pruebas y de operación posterior.

RNF-7: Todo el sistema deberá estar complemente documentado y, en concreto, cada uno de los componentes de software que forman parte de la solución propuesta, tanto en el código fuente como en los manuales de administración y de usuario.

RNF-8: El acceso al Sistema debe estar restringido por el uso de claves asignadas a cada uno de los usuarios. Sólo podrán ingresar al Sistema las personas que estén registradas. Los usuarios serán clasificados en varios tipos de usuarios (o roles) con acceso a las opciones de trabajo definidas para cada rol.

RNF-9: El control de acceso implementado debe permitir asignar los perfiles para cada uno de los roles identificados.

RNF-10: El sistema debe validar automáticamente la información contenida en los formularios de ingreso. En el proceso de validación de la información, se deben tener en cuenta aspectos tales como obligatoriedad de campos, longitud de caracteres permitida por campo, manejo de tipos de datos, etc.

RNF-11: La solución debe ser 100% Web y todas las operaciones deben realizarse desde un navegador.

RNF-12: La solución debe operar de manera independiente del navegador que se utilice.

RNF-13: La solución debe tener interfaces gráficas de administración y de operación en idioma español y gallego.

RNF-14: La aplicación móvil detectará automáticamente la provincia en la que se encuentra el usuario.

RNF-15: Los backup´s deben ser responsabilidad del administrador del sistema quien deberá crearlos, almacenarlos y recuperar la información en el caso que se pierda información. Se deberá realizar una copia diaria de la información y borrar las copias que excedan de una semana.

RNF-16: La LOPD dice que en los sitos web con operación monetarias hay que guardar los datos de los usuarios por lo menos durante 4 años. Por lo que las empresa, usuarios y ofertas que sean borradas, ya sea por cuenta propia o del administrador, deberán ser guardadas en la base de datos durante, al menos, 4 años.

4 ANÁLISIS Y DISEÑO DEL SISTEMA

Para realizar el análisis se ha utilizado el Análisis Orientado a Objetos (AOO) que se define como "un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema". Los objetos son entidades tangibles que muestran un comportamiento bien definido. Se ha utilizado este tipo de análisis ya que el lenguaje seleccionado para la implementación del sistema es Java, que es un lenguaje orientado a objetos, y porque se ha considerado esta metodología como la más adecuada para este tipo de problema.

Todo esto quiere decir que el análisis orientado a objetos parte de entidades tangibles halladas en el problema; Tales entidades varían dependiendo de los diversos casos prácticos, pero en todos los casos son elementos reales que toman parte del problema de forma directa. El Diseño Orientado a Objetos (DOO) "es el método que lleva a una descomposición Orientada a Objetos. Aplicando DOO, se crea software resistente al cambio. Se logra un mayor nivel de confianza en la corrección del software a través de la división inteligente de su espacio de estados. En última instancia, se reducen los riesgos inherentes al desarrollo de sistemas.

Los modelos del diseño orientado a objetos reflejan la importancia de plasmar explícitamente las jerarquías de clases y objetos del sistema que se diseña. Estos modelos cubren también el espectro de las decisiones de diseño relevantes que hay que considerar en el desarrollo de un sistema complejo, y así animan a construir implantaciones que posean los atributos de los sistemas complejos bien formados.

Vamos a seleccionar un requisito del apartado anterior y vamos a seguir su evolución a través de todos las etapas siguientes para ilustrar la metodología de diseño empleada y su notación. El requisito elegido es el *RF-25 Un usuario se identificará en el sistema mediante su dirección de correo electrónica y su clave.*

4.1 Análisis del sistema

En esta etapa se logra claridad sobre lo que desea el usuario y la forma en la cual se va a presentar la solución que se está buscando. Se examinan los requisitos desde la perspectiva de los objetos y clases del dominio del problema.

4.1.1 Casos de uso

El modelo de casos de uso describe un sistema en base a sus distintas formas de utilización, cada una de las cuales es conocida como un caso de uso. Cada caso de uso o flujo se compone de una secuencia de eventos iniciada por el usuario. Dado que los casos de uso describen el sistema a desarrollar, cambios en los requisitos significarán cambios en los casos de uso.

Los diagramas de casos de uso muestran las operaciones que puede realizar cada actor. Para ello se define el concepto de actor, correspondiente al tipo de usuario que está involucrado en la utilización de un sistema, siendo el actor una entidad externa al propio sistema. Los actores que participaran en el sistema son:

Actor: S Sistema	
Descripción	*El sistema de ofertas*

Actor: ACT-1 Usuario No registrado	
Descripción	*Actor antes de acceder en la aplicación*

Actor: ACT-2 Usuario registrado	
Descripción	*Actor después de acceder como usuario*

Actor: ACT-3 Empresa registrada	
Descripción	*Actor después de acceder como empresa*

Actor: ACT-4 Institución registrada	
Descripción	*Actor después de acceder como institución*

Actor: ACT-5 Administrador	
Descripción	*Actor después de acceder como administrador*

Actor: ACT-6 Facebook

Descripción	*Red social externa*

Actor: ACT-7 Twitter	
Descripción	*Red social externa*

Actor: ACT-8 Google maps	
Descripción	*Gestor de mapas externo*

Tras haber definido los actores del sistema, se identifican los casos de uso del sistema a partir de los requisitos. Éstos junto con los actores representan los dos elementos básicos del diagrama de casos de uso. Para la mejor comprensión del diagrama de casos de uso, se ha diseñado un diagrama por cada actor del sistema. El diagrama de casos de uso del sistema es el siguiente:

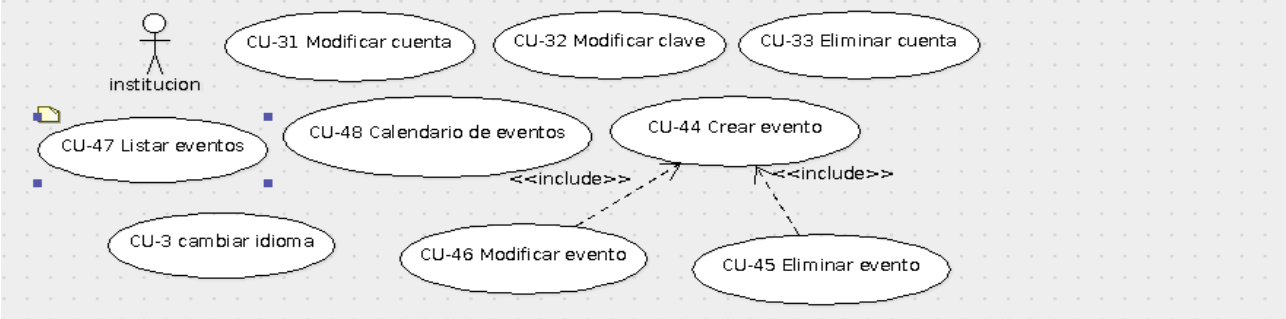


Ilustración 3: Diagrama de casos de uso de una institución

Ilustración 4: Diagrama de casos de uso de un usuario

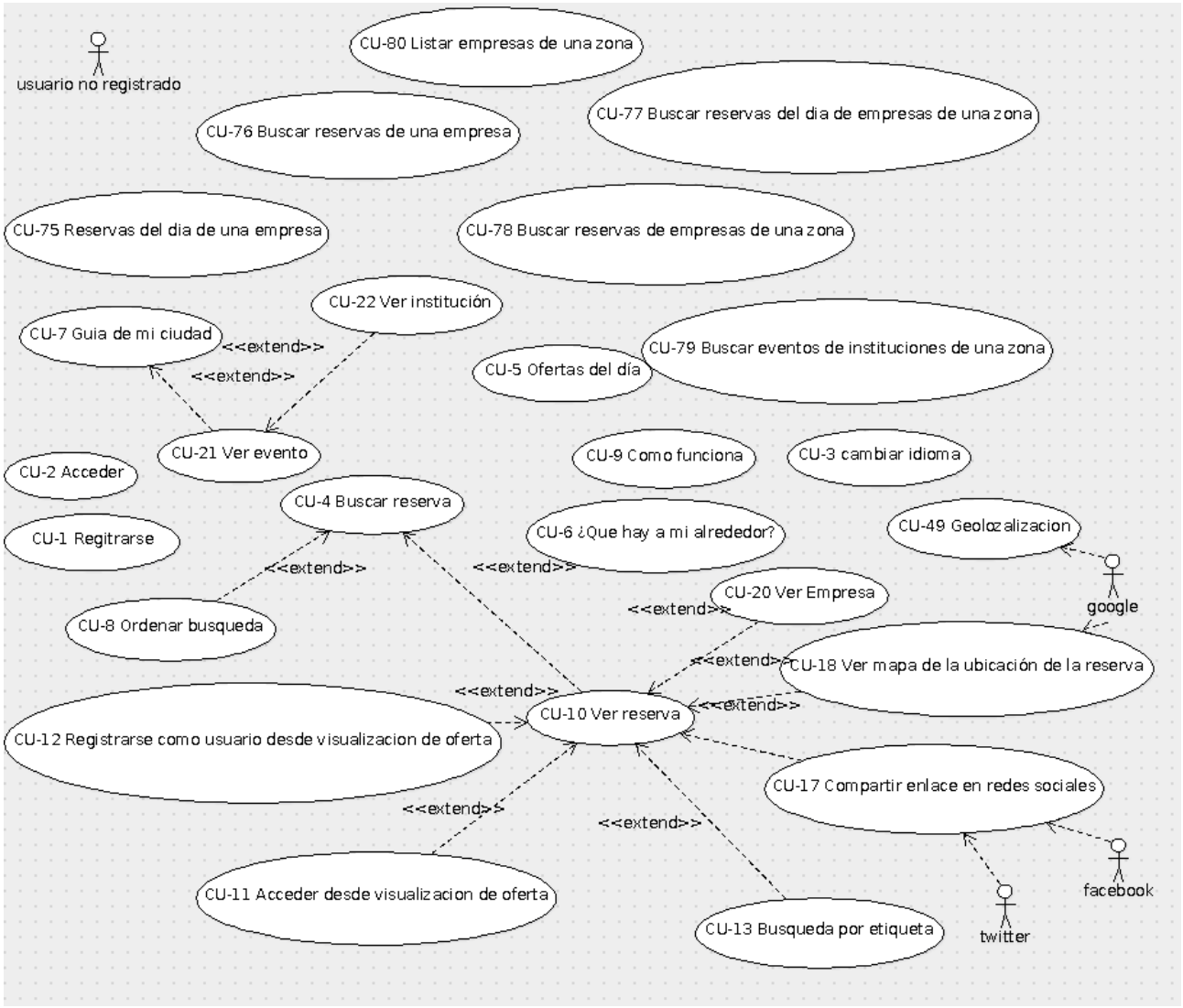


Ilustración 5: Diagrama de casos de uso de un usuario no registrado

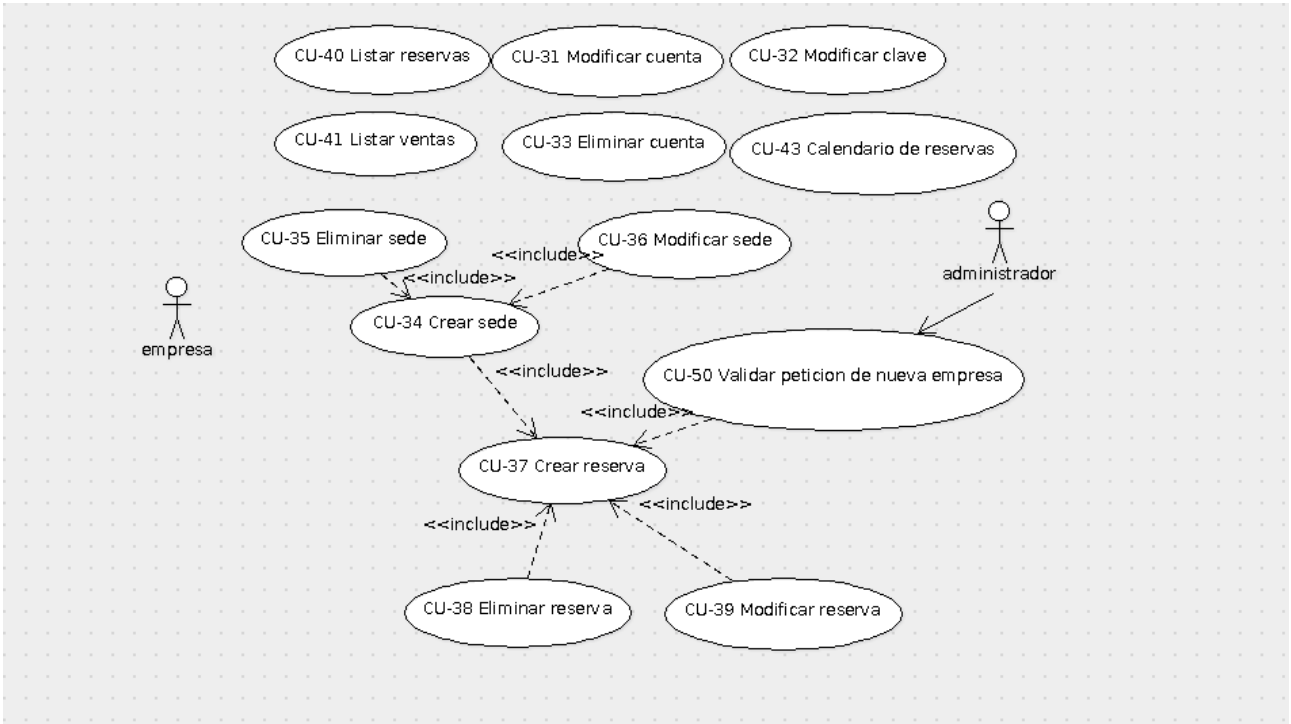


Ilustración 6: Diagrama de casos de uso de una empresa

Ilustración 7: Diagrama de casos de uso de un administrador

Una vez que hemos identificado todos los casos de uso, hay que definirlos en alto nivel y para ello se ha utilizado la siguiente plantilla:

Caso de Uso: CU-<Número Secuencia> Nombre	
Descripción	*< Breve descripción de la funcionalidad del caso de uso>*
Actores	*< Lista de actores que participan en la realización del caso de uso>*
Secuencia Normal	*< En este apartado se describe el escenario principal del caso de uso mediante una secuencia normal de pasos o acciones necesaria para cumplir con éxito la funcionalidad del caso de uso>*
Secuencia(s) Alternativas(s)	*< En este apartado se describen uno o más escenarios alternativos del caso de uso que representan situaciones distintas a la normal (p.e. ha ocurrido un error, y describe cómo tiene que responder el sistema ante tal situación)>*
Precondiciones	*< Condiciones que se deben cumplir antes de ejecutar el caso de uso>*
Postcondiciones	*< Condiciones que se deben cumplir al finalizar el caso de uso>*

Tabla 2: Plantilla para definir casos de uso

En el caso concreto del requisito *RF-25 Un usuario se identificará en el sistema mediante su dirección de correo electrónica y su clave*, buscamos en la *matriz de trazabilidad casos de uso – requisitos* del **anexo A** el caso de uso relacionado con el requisito y vemos que es *CU-2 identificarfse.*

Caso de Uso: CU-2 Identificarse	
Descripción	El actor puede identificarse sistema mediante su dirección de correo electrónico y su clave. Accederá al apartado gráfico correspondiente al tipo de usuario al que pertenezca.
Actores	*S, ACT-1*
Precondiciones	El actor dispone de conexión a internet. El sistema funciona correctamente. El actor se encuentra en el sistema pero aun no ha accedido. El actor está registrado. El actor se encuentra en la página de acceso.(ilustración 8).
Secuencia Normal	1. ACT: introduce su dirección de correo electrónico y su contraseña y le da "aceptar". 2. S: Verifica en la base de datos que existe una cuenta de algún tipo de usuario con ese dirección de correo electrónico y esa contraseña y muestra la pantalla principal para ese tipo de usuario(ilustración 9).
Secuencia(s) Alternativas(s)	2a. S: No existe el dirección de correo electrónico en la base de datos. Muestra un mensaje de error informando de que no existe la cuenta (ilustración 10). 2b. S: Contraseña errónea. Muestra un mensaje de error informando que la contraseña es errónea (ilustración 10). 2c. S:Falla al insertar en la base de datos. Muestra un mensaje informando de dicho error(ilustración 11).
Postcondiciones	El actor ha accedido al apartado gráfico del sistema correspondiente al tipo de usuario al que pertenece. El sistema no sufre incidencias y su funcionamiento es adecuado.

Tabla 3: Ejemplo de casos de uso

El resto de casos de uso se encuentra desarrollado en detalle en el **anexo A**.

4.1.2 Diseño de interfaces

Dado que el modelo de casos de uso está motivado y enfocado principalmente hacia los sistemas de información donde los usuarios juegan un papel primordial, es importante ya relacionarse con las interfaces a ser diseñadas en el sistema. Estas interfaces sirven para apoyar de mejor manera la descripción de los casos de uso además de servir de base para prototipos iniciales.

El modelo de interfaces describe la presentación de información entre los actores y el sistema. Se especifica en detalle como se verán las interfaces de usuario al ejecutar cada uno de los casos de uso. Para el diseño ha sido esencial que las interfaces reflejen la visión lógica del sistema ya que es uno de los principios fundamentales del diseño de interfaces humanas, donde debe existir consistencia entre la imagen conceptual del usuario y el comportamiento real del sistema.

Para diseñar el interfaz, se ha realizado un diseño de cada pantalla que debe presentar el sistema siguiendo la secuencia de pasos de cada caso de uso. Para el diseño de las pantallas se ha usado el programa Balsamiq.

Como hemos visto en el apartado anterior, en *CU-2 identificarse* partimos de la pantalla principal (ilustración 8) y, siguiendo los la secuencia normal de pasos llegamos a la pantalla principal para usuarios (ilustración 9). El diseño de estas pantallas es el siguiente:

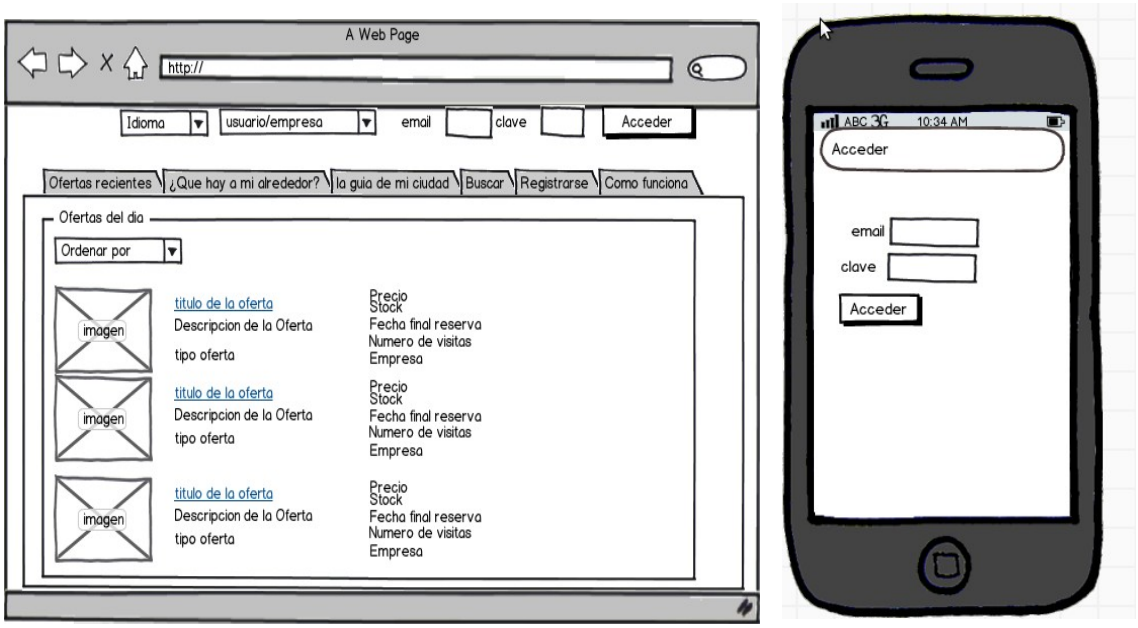


Ilustración 8: Pantalla principal

La *Ilustración 8* muestra todas las funcionalidades que tiene un usuario que aun no ha accedido al sistema desde el apartado de internet y que no son una extensión de otra funcionalidad. Estas funcionalidades son cambiar el idioma, acceder, ofertas del día, ¿que hay a mi alrededor?, la guía de mi ciudad, buscar reservas, registrarse y conocer el funcionamiento del sistema.

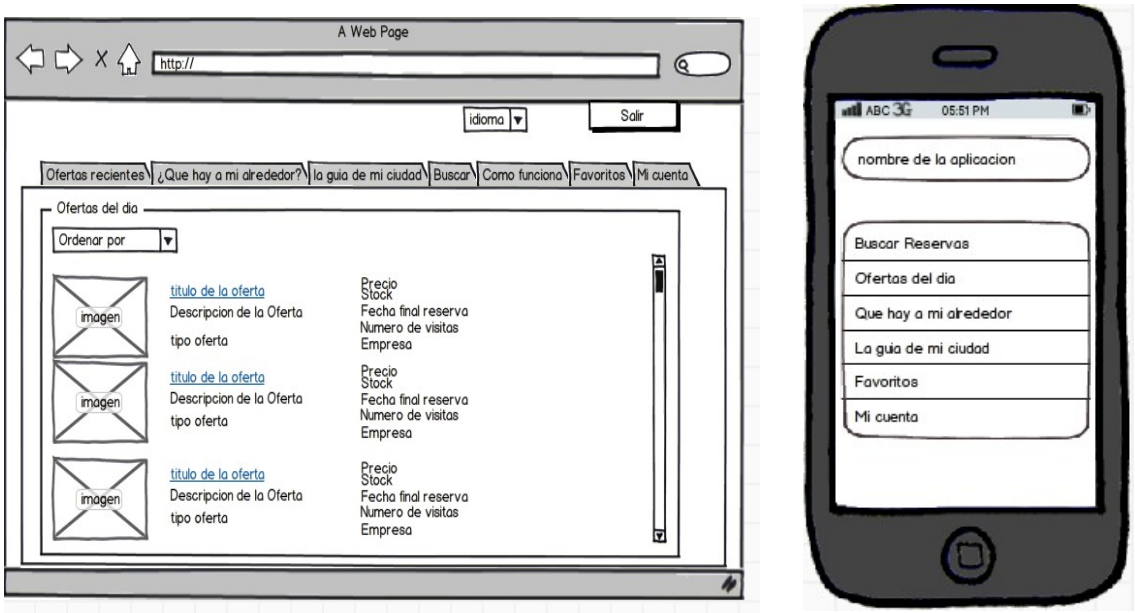


Ilustración 9: Pantalla principal para usuarios

La *Ilustración 9* muestra todas las funcionalidades que tiene un usuario que ha accedido como tipo de usuario *usuario* al sistema desde el apartado de internet y desde el apartado móvil que no son una extensión de otra funcionalidad. Estas funcionalidades son cambiar el idioma, salir, ofertas del día, ¿que hay a mi alrededor?, la guía de mi ciudad, buscar reservas, favoritos, mi cuenta y conocer el funcionamiento del sistema.

Si ocurriera un error durante la secuencia normal, se mostrarían las pantallas de error (ilustración 10 y 11). El diseño de las pantallas de error es el siguiente:

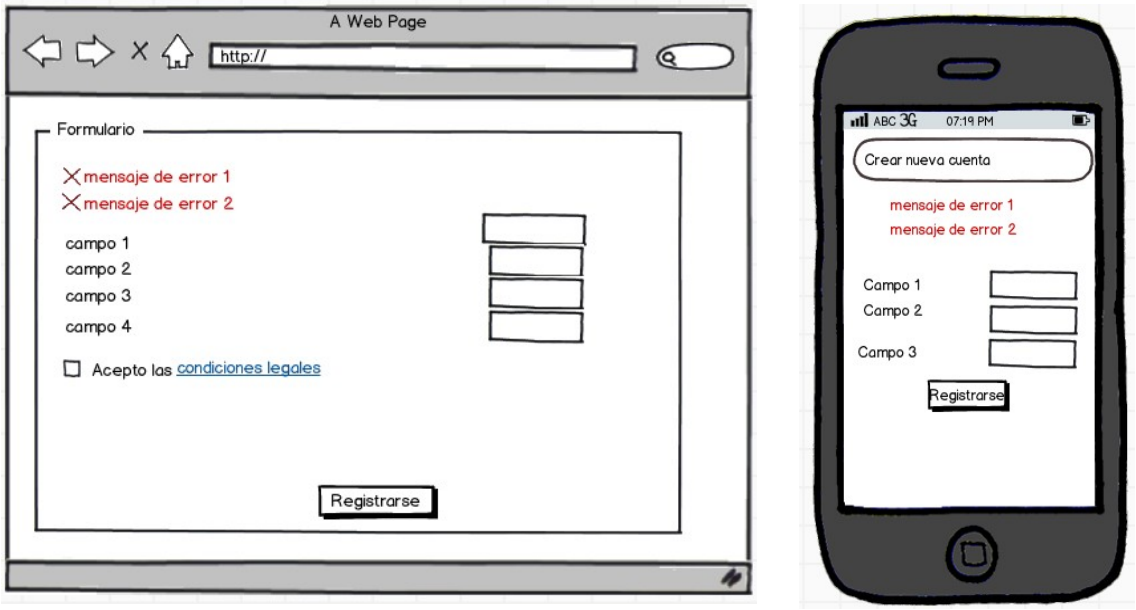


Ilustración 10: Error en formulario

La *Ilustración 10* muestra mensajes de error según los requisitos funcionales de validación de datos y los requisitos no funcionales RNF-5 y RNF-10.

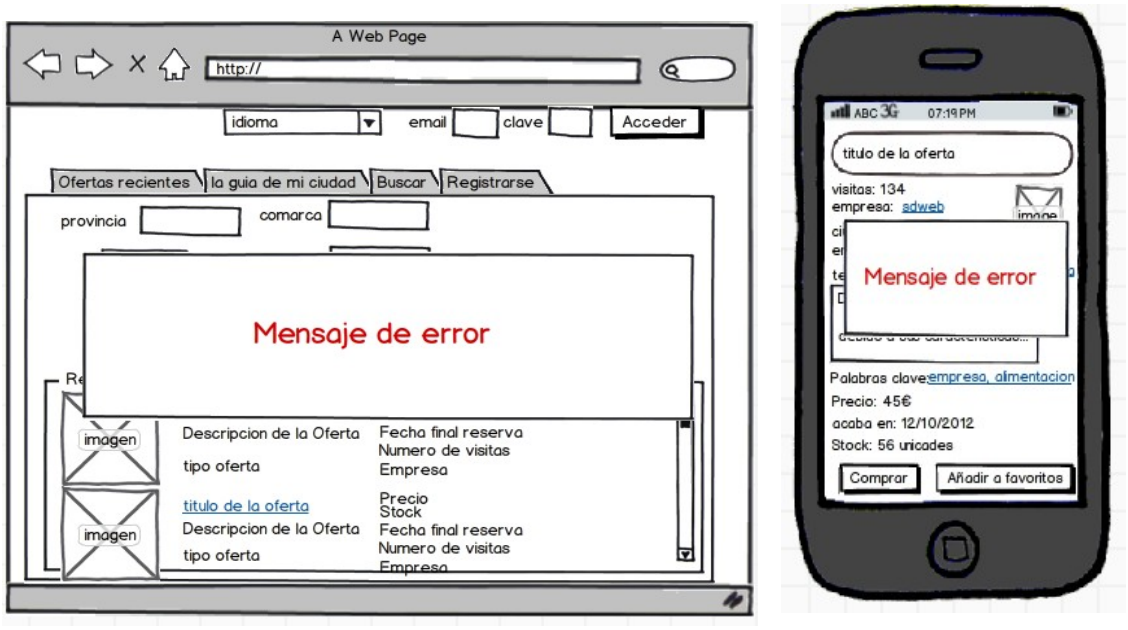


Ilustración 11: Error en base de datos

La *Ilustración 11* muestra mensajes de error según el requisitos no funcional RNF-5.

El resto de diseños de los interfaces se encuentran en el **anexo B**.

4.1.3 Diagramas de secuencia

Para describir la información de entrada y salida de cada caso de uso se utilizan los diagramas de secuencia. Un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian, ordenados según su secuencia en el tiempo. Un diagrama de secuencia muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos.

Vaadin implementa el patrón MVC (Modelo-Vista-Controlador), una arquitectura que busca reducir el acoplamiento, dividiendo las responsabilidades en 3 capas claramente diferenciadas:

- El **modelo,** que a su vez está divido en dos capas: la capa que hace referencia a los datos que maneja la aplicación y la **lógica** de negocio que opera sobre ellos, y la capa de **persistencia** (DAO) que es la que interactúa con la base de datos.
- La **vista**, encargada de generar la interfaz con la que la aplicación interacciona con el usuario.
- El **controlador**, que comunica la vista y el modelo, respondiendo a eventos generados por el usuario en la vista, invocando cambios en el modelo, y devolviendo a la vista la información del modelo necesaria para que pueda generar la respuesta adecuada para el usuario.

Para el diseño de los diagramas de secuencia se va a tener en cuenta esta arquitectura ya que presenta varias ventajas como la organización del código, la reutilización y la flexibilidad.

Un diagrama de secuencia siguiendo esta arquitectura presenta el siguiente esquema.

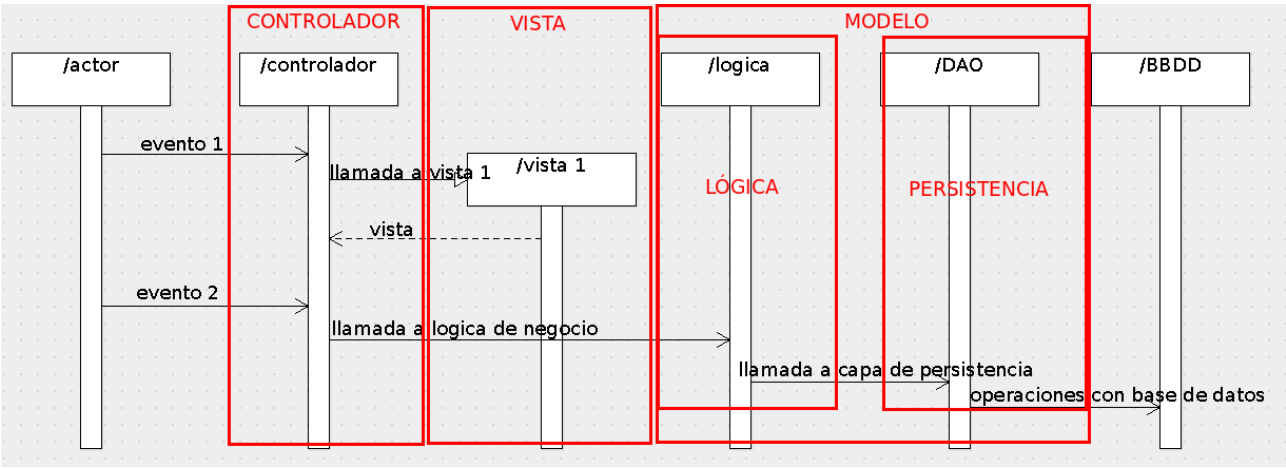


Ilustración 12: Esquema de un diagrama de secuencia

Para construir un Diagrama de Secuencia del Sistema para el curso típico de eventos de un caso de uso, se dibuja una línea para el actor que opera directamente con el sistema y otra para la clase controlador. En el diagrama anterior corresponderían con *actor* y *controlador*.

Después, partiendo del texto del curso típico de eventos del caso de uso, las acciones realizadas por el actor hay que identificarlas como eventos (externos) del sistema y se representan en el diagrama como un mensaje desde el actor hacia el controlador. En el diagrama anterior corresponderían con los mensajes *evento1* y *evento2*.

Por otro lado, las acciones realizadas por el sistema pueden ser de dos tipos: mostrar una ventana o realizar una operación. Cuando una acción del sistema es del tipo mostrar una vista, se crea una clase vista que se añade al diagrama (en el diagrama anterior *vista1*) y se manda un mensaje de creación desde el controlador hasta la clase. En el diagrama anterior se correspondería con el mensaje *llamada a vista 1*.

Si una acción del sistema es del tipo operación, entonces se añade al diagrama la clase relacionada con la operación (en el diagrama anterior *lógica*) y se añade al diagrama un mensaje desde el controlador hacia la clase con una llamada a una función de la clase. En el diagrama anterior se correspondería con el mensaje *llamada a lógica de negocio.*

Si la operación tiene que usar la base de datos, se añade al diagrama la clase *DAO* y un mensaje desde el objeto relacionado con la operación hacia la clase *DAO* con una llamada a una función de la clase *DAO*. En el diagrama anterior se correspondería con el mensaje *llamada a capa de persistencia.*

Por ejemplo, si una acción del sistema es “Buscar todas las empresas en la base de datos” se

añadiría al diagrama la clase *empresa* y un mensaje con una función, por ejemplo, *buscarTodasOfertas()* desde el controlador hasta *empresa.* Luego se añadiría al diagrama la clase *DAO* y un mensaje con una función también llamada, por ejemplo, *buscarTodasOfertas()* desde el objeto *empresa* hasta un objeto *DAO. S*e ha creado una clase *lógica* para añadir las funciones que no tiene relación con ninguna clase o con mas de una,

Para ilustrar esta metodología, el diagrama de secuencia correspondiente al caso de uso utilizado como ejemplo, el *CU-2 identificarse,* es el siguiente:

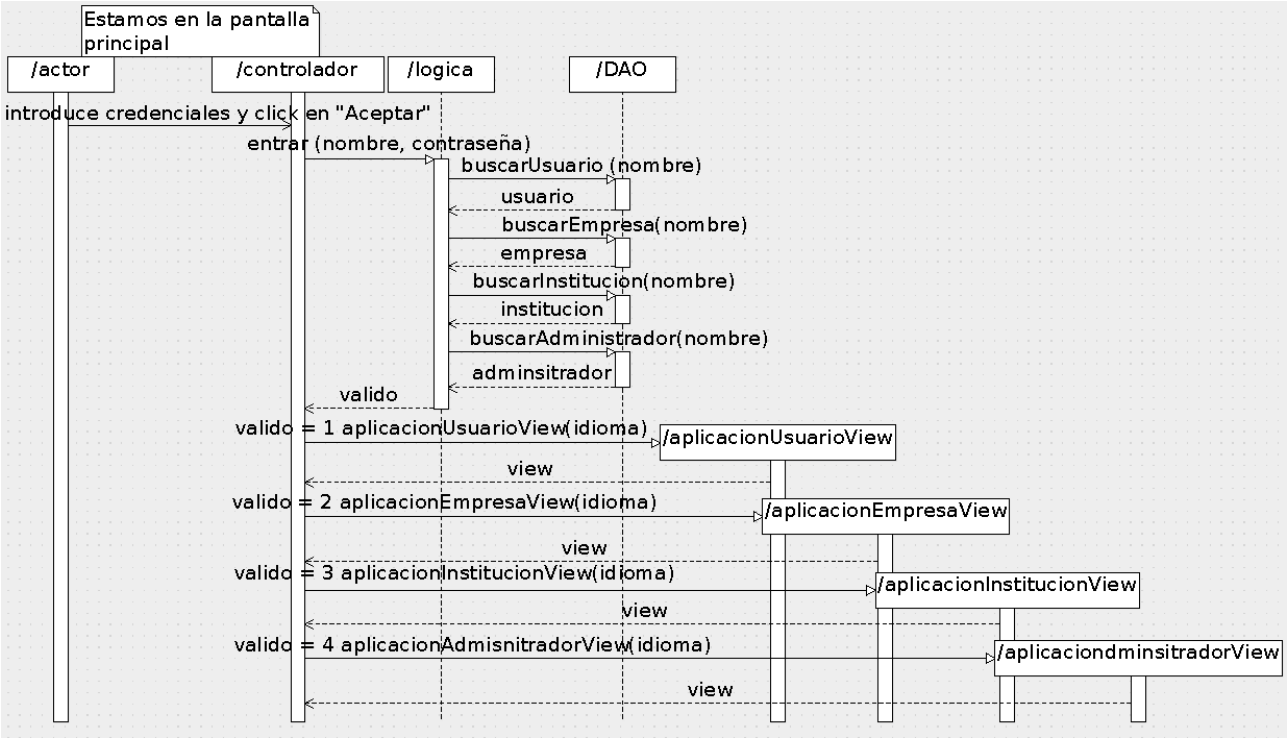


Ilustración 13: diagrama de secuencia del CU-2 identificarse

El actor introduce sus credenciales y pulsa el botón "Aceptar", generando un evento en la pantalla principal que se captura en el controlador. El controlador al capturar el evento llama a la función *entrar(nombre,contraseña)* de la clase lógica que llama a las funciones *buscarUsuario(nombre), buscarEmpresa(nombre), buscarinstitución(nombre) y buscarADministrador(nombre)* de la clase DAO. Una vez terminadas todas estas acciones, el controlador llama a la vista correspondiente.

En el **anexo C** se pueden consultar el resto de los diagramas de secuencia.

4.2 Diseño del sistema

Tal y como se ha puesto de manifiesto de forma precisa en los requisitos, la central de ofertas implementará un sistema de ofertas nacional accesible desde un navegador o desde un dispositivo móvil. Ademas la central de ofertas dispondrá de un sistema de ofertas especifico de un Ayuntamiento.

Dado que en el sistema se aprecian tres apartados claramente diferenciados, se ha decidido implementar la central de ofertas en base a tres aplicaciones distintas pero con elementos comunes como veremos en los capítulos siguientes. Una aplicación web que implementará un sistema de ofertas que permita que cualquier empresa particular o institución, a nivel nacional, una vez dado de alta en el sistema, pueda ofertar sus productos, servicios y eventos para que el consumidor pueda informarse o comprar desde internet. Otra aplicación para que el consumidor pueda informarse o comprar desde un teléfono móvil y, finalmente, una tercera aplicación que permita la compra-venta de productos y servicios y visualización de eventos y noticias de interés, de las empresas e instituciones que pertenezcan al entorno de un Ayuntamiento.

En este apartado se va a mostrar como se ha definido la subdivisión en aplicaciones del sistema.

4.2.1 Diagrama de clases

Una vez que se han diseñado los diagramas de secuencia de todos los casos de uso, se pasa a diseñar el diagrama de clases para presentar las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones.

Para diseñar el diagrama de clases, primero se han identificado todas las clases participantes en los diagramas de secuencia y se han organizado en paquetes o módulos. Para cada clase se han añadido los atributos, definidos en los requisitos y los métodos, según aparecen en los diagramas de secuencia. Después se ha añadido información de tipo a los atributos y métodos. Para definir una clase se ha utilizado el siguiente esquema.

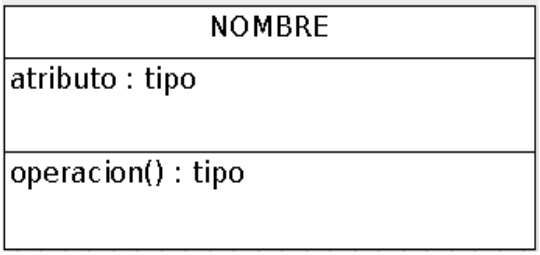


Ilustración 14: Esquema de definición una clase en un diagrama de clases

Las clases que componen el paquete correspondiente al **modelo** de datos y sus relaciones se muestran en el diagrama siguiente. La descripción de los atributos y funciones de cada clase se encuentran detalladas en el **anexo D.**

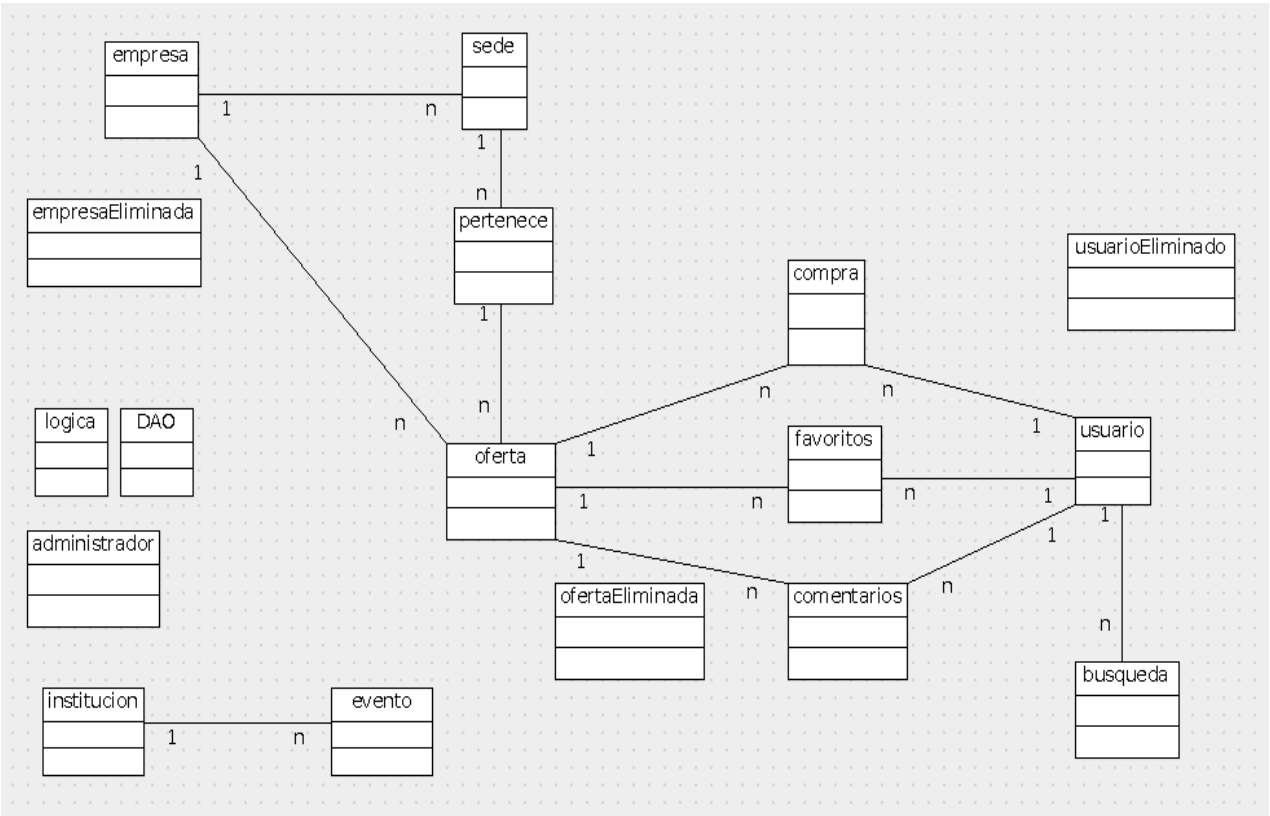


Ilustración 15: Relación entre las clases que componen el paquete modelo

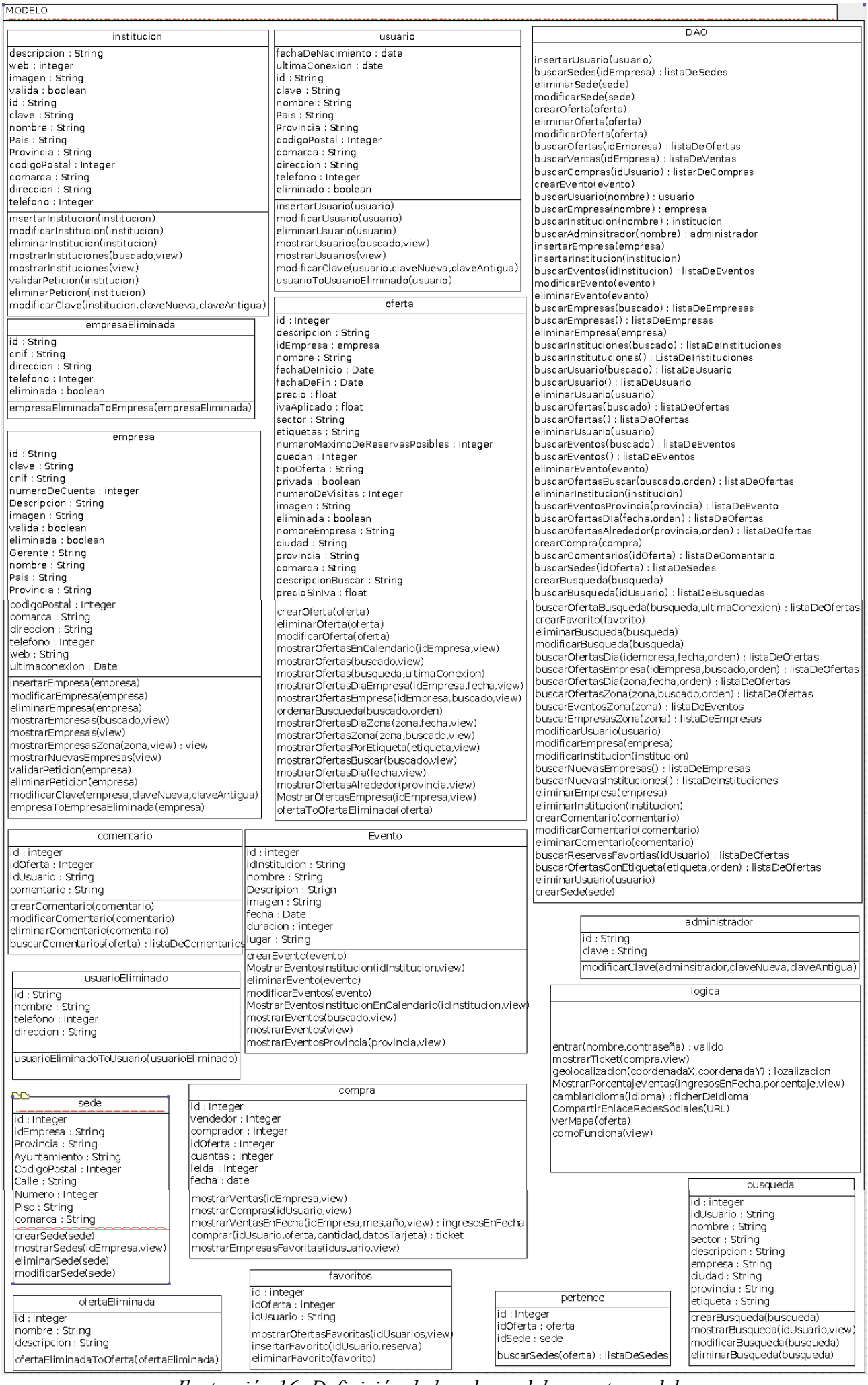


Ilustración 16: Definición de las clases del paquete modelo

Los elementos de un interfaz que generan eventos, en Vaadin, han de definirse como variables globales de la clase que crea el interfaz . De esta manera, los atributos de las clases que componen la vista del sistema han sido definidos examinando el diseño de interfaces y seleccionando aquellos elementos que generan eventos como botones, tablas, cajas de selección, etc. Además el propio interfaz ha de ser definido como variable global de la clase.

Las clases que componen la **vista** han sido agrupadas en paquetes por aplicación y por tipo de usuario con el que interaccionará. Un paquete contiene las clases que generan los interfaces que usara un tipo de usuario en una aplicación.

De esta manera, los diferentes paquetes creados son:

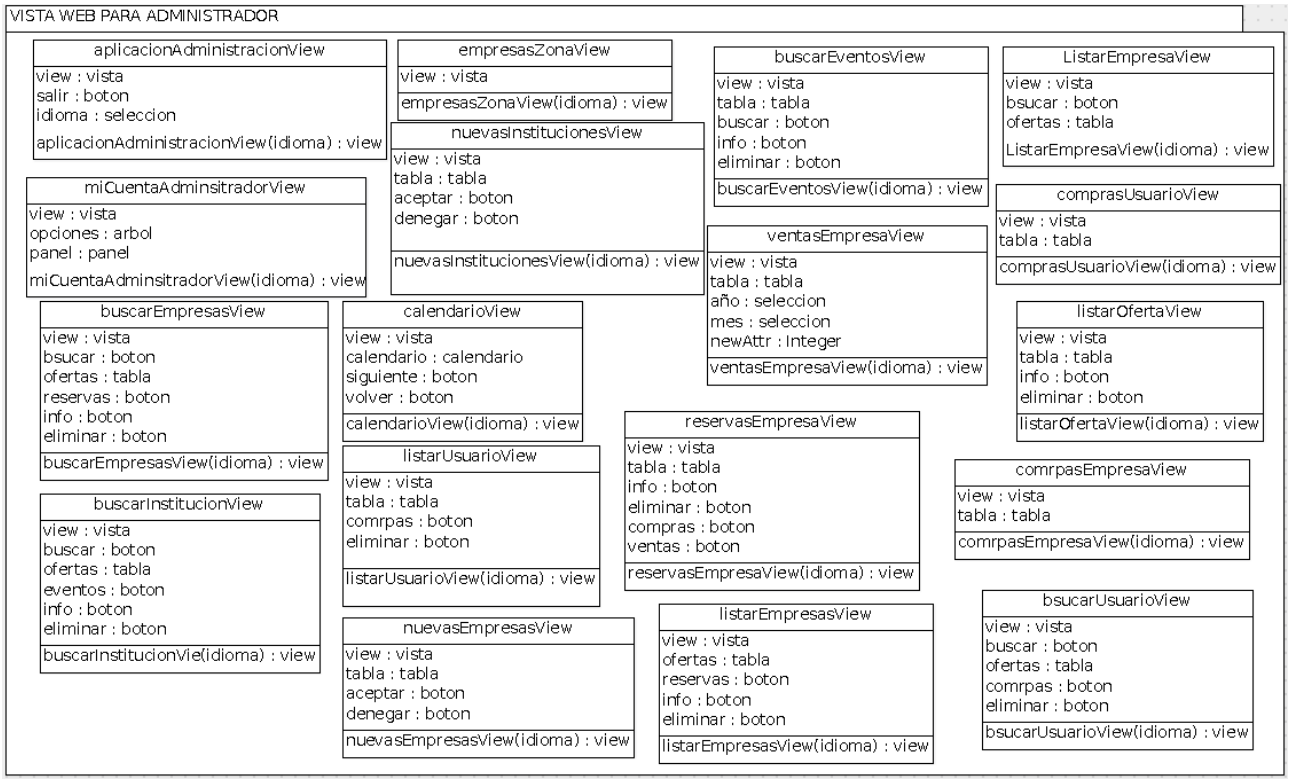


Ilustración 17: Clases del paquete vista web para un administrador

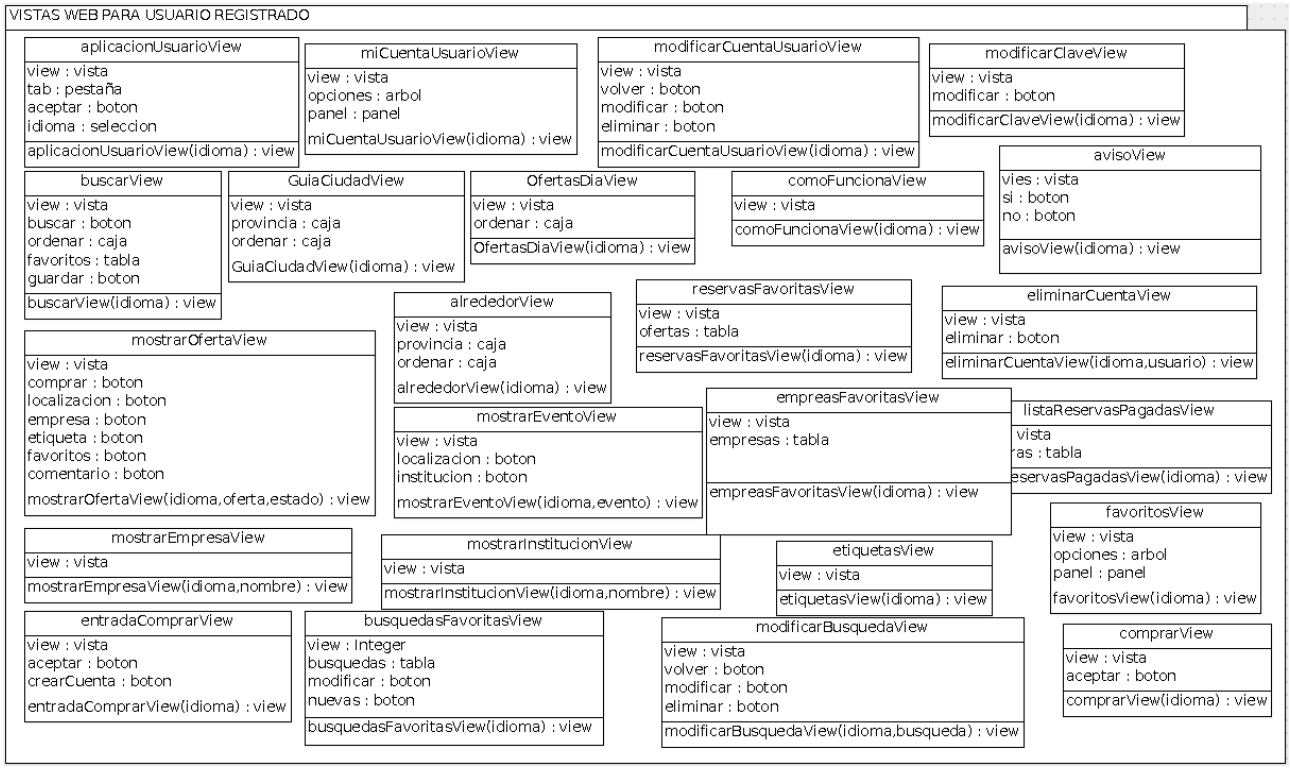


Ilustración 18: Clases del paquete vista web para un usuario

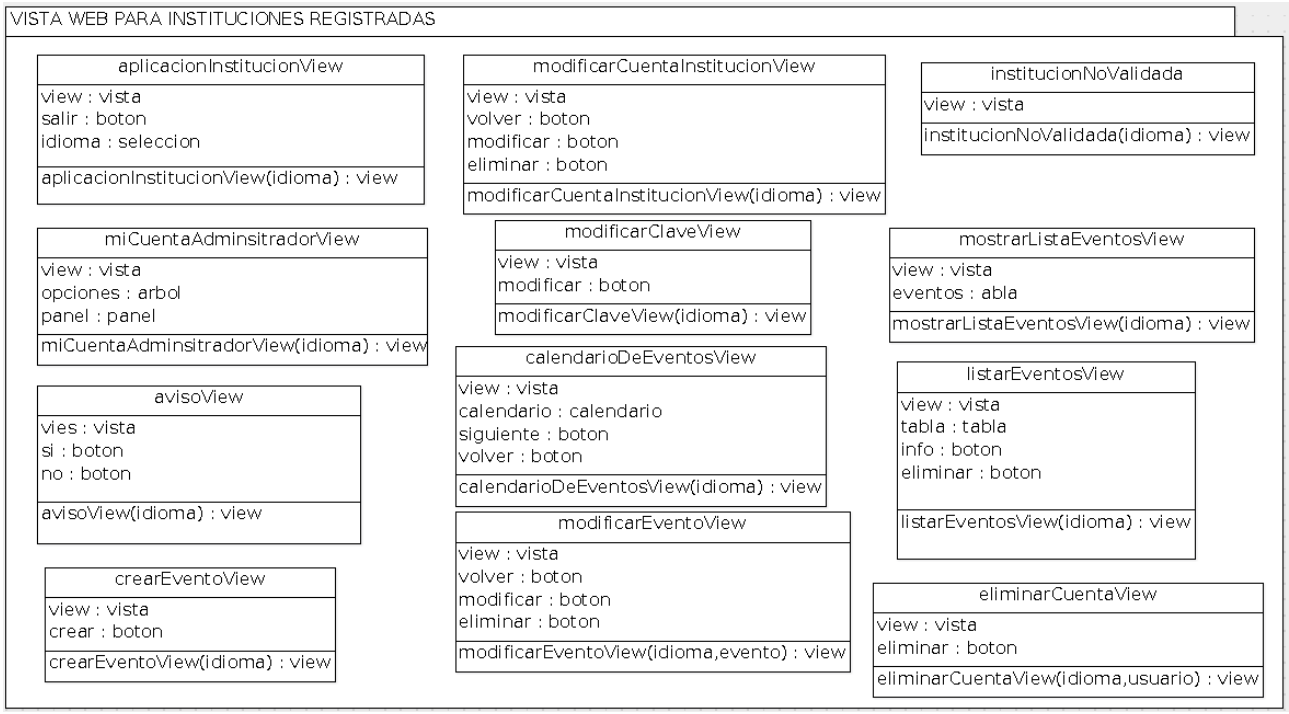


Ilustración 19: Clases del paquete vista web para una institución

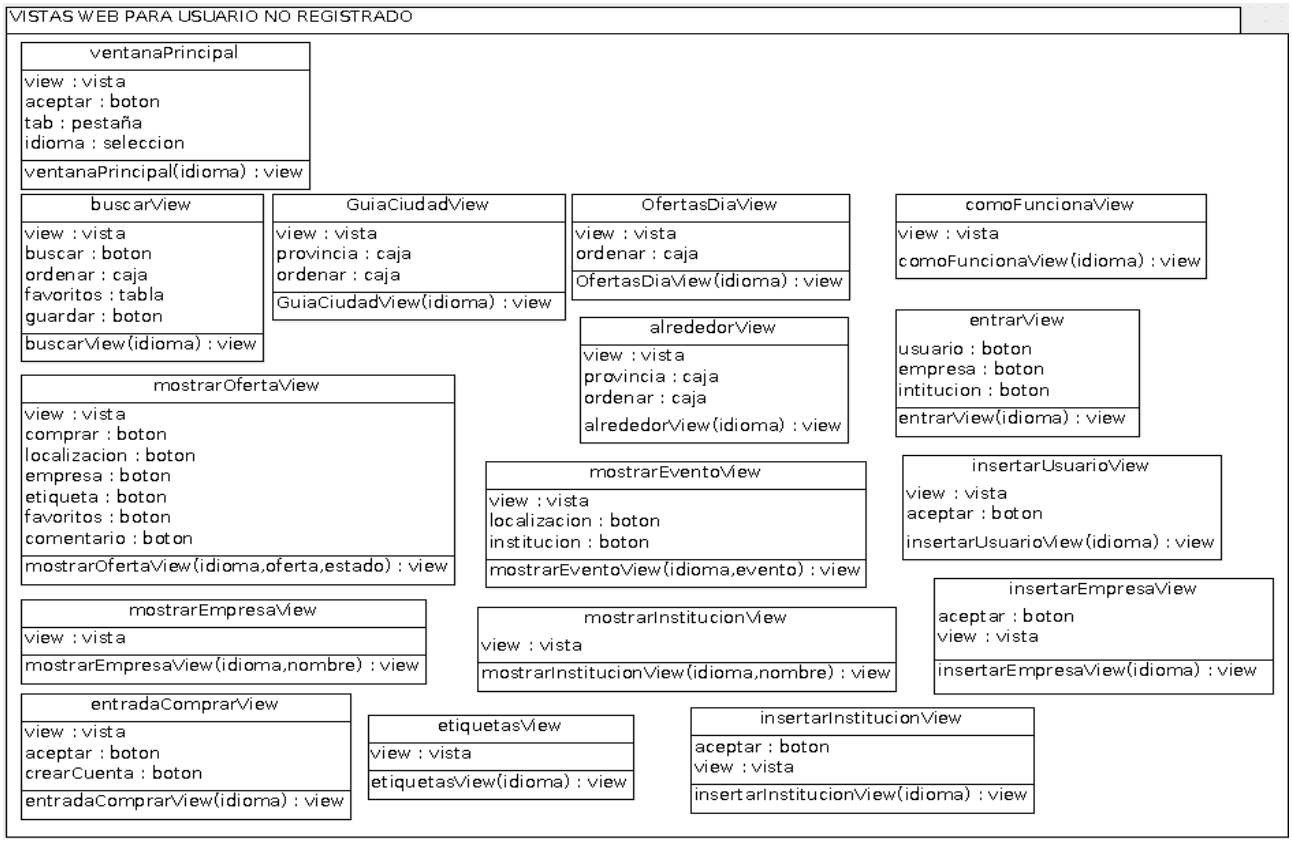


Ilustración 20: Clases del paquete vista web para un usuario no registrado

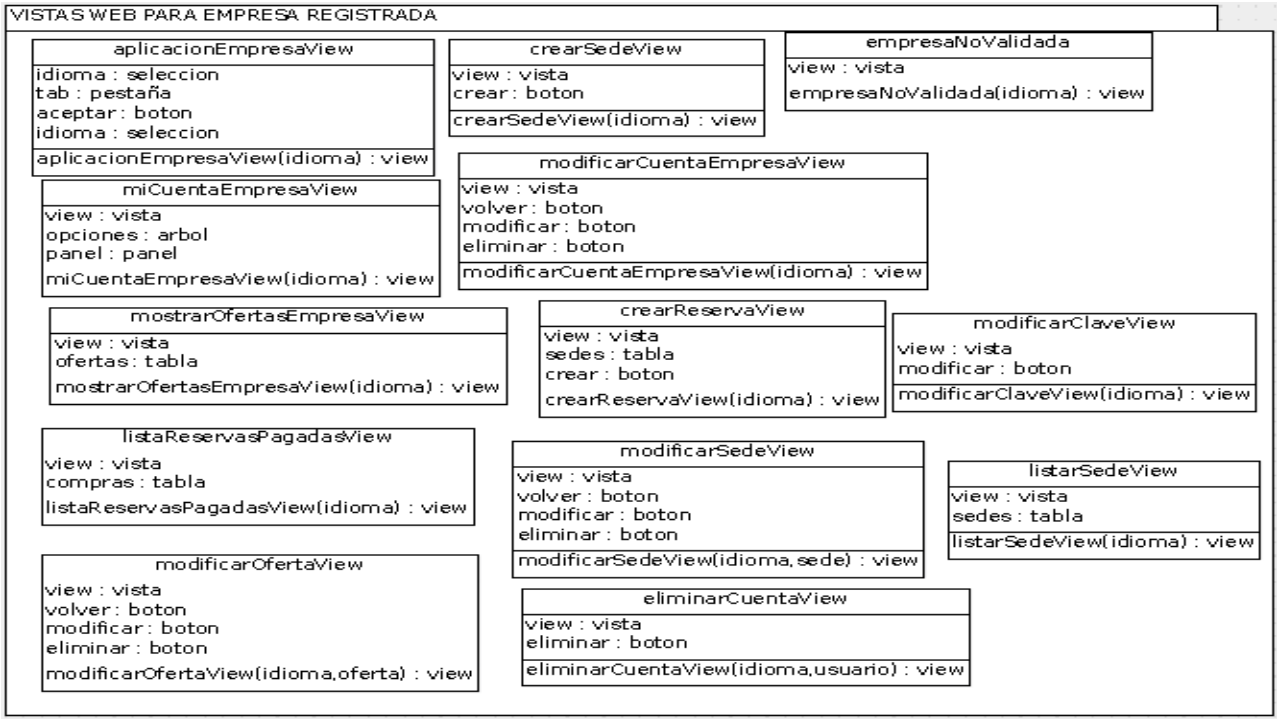


Ilustración 21: Clases del paquete vista web para una empresa

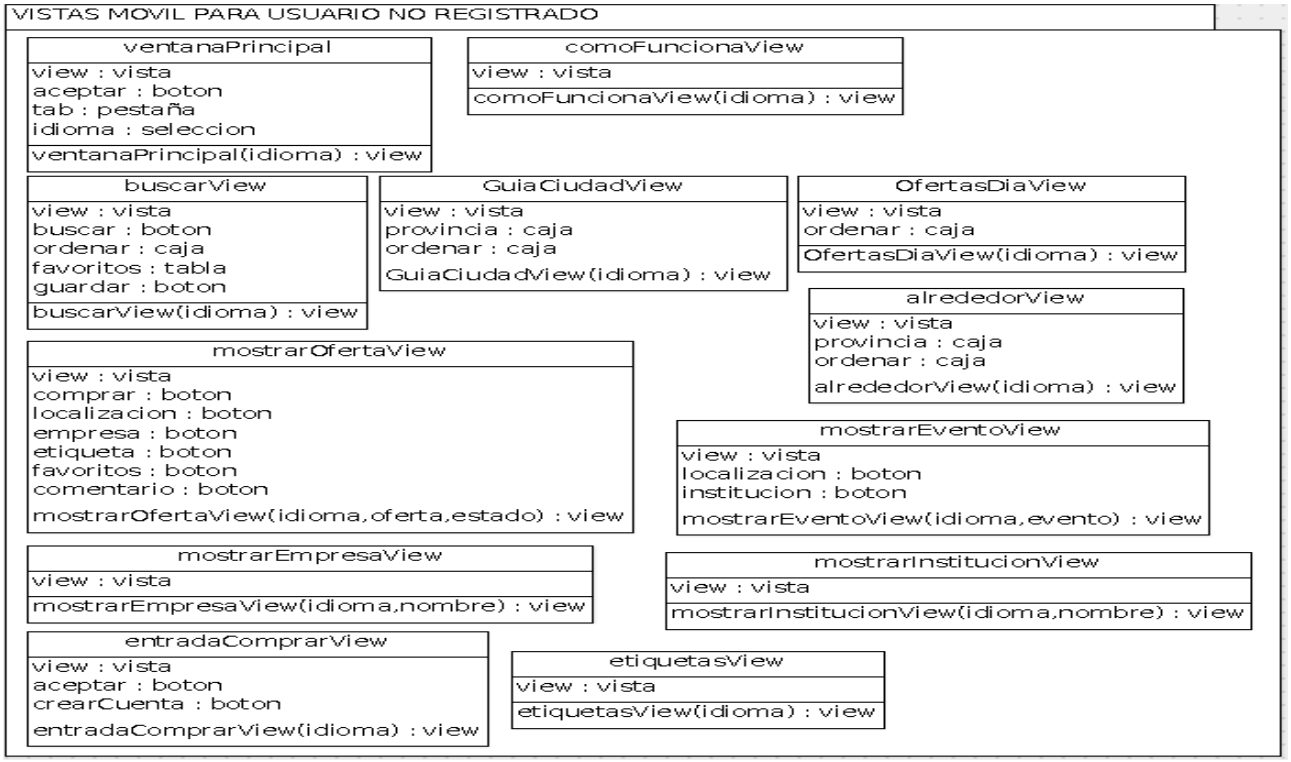


Ilustración 22: Clases del paquete vista móvil para un usuario no registrado

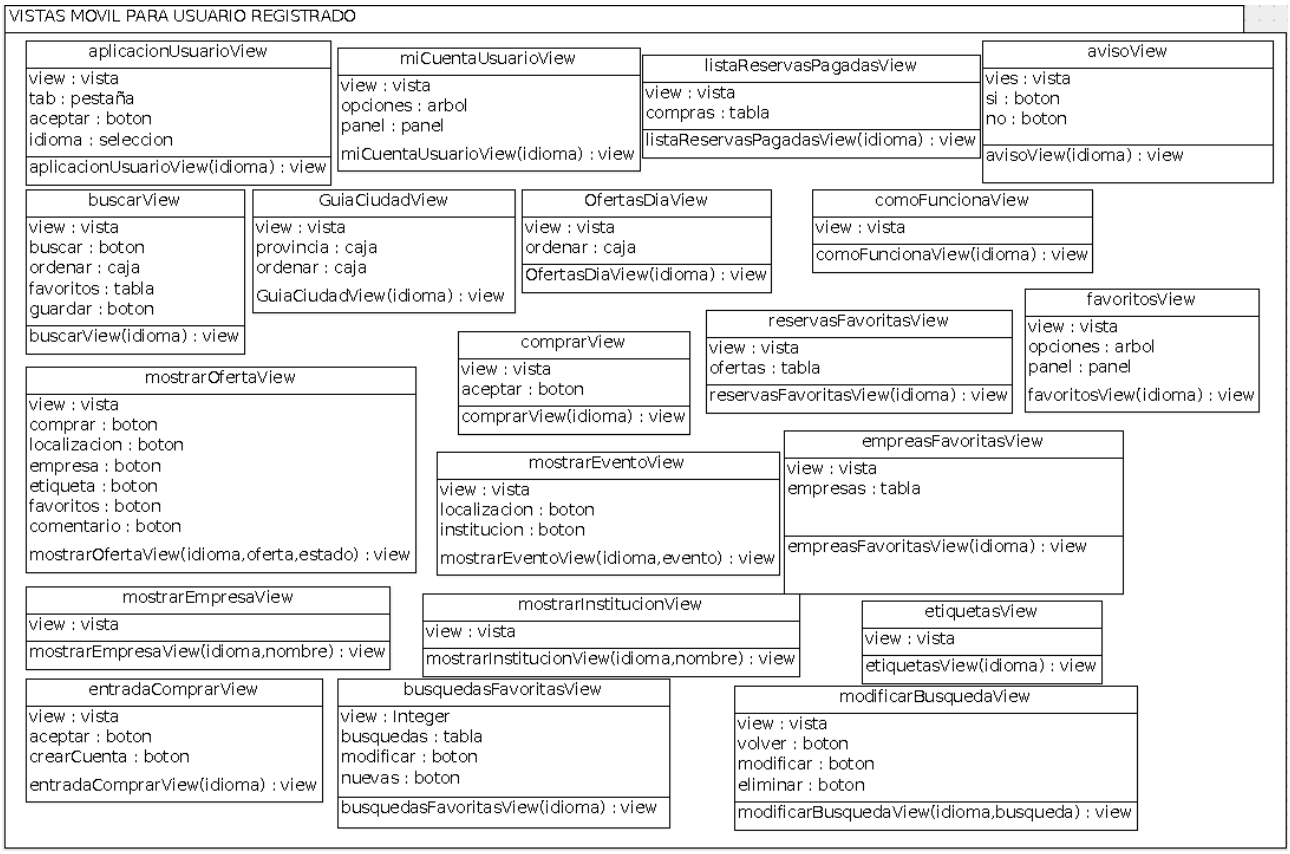


Ilustración 23: Clases del paquete vista móvil para un usuario

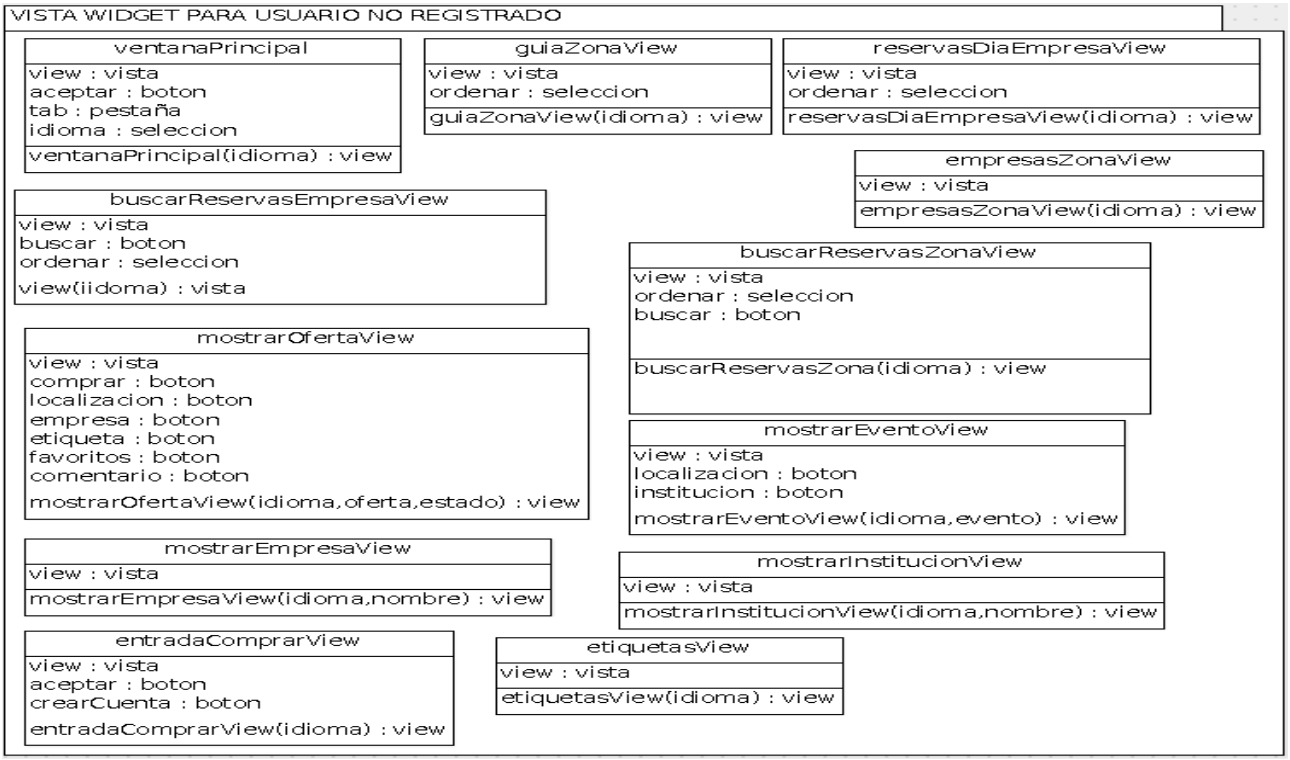


Ilustración 24: Clases del paquete vista widget para un usuario no registrado

Ilustración 25: Clases del paquete vista widget para un usuario

Las clases de un paquete de la vista no tienen relación entre ellas, ya que se comunican a través del controlador.

Vaadin opera de tal manera que el paquete **controlador** sólo se compone de la clase *controlador*. Esta clase será la encargada de llamar a las clases de la vista que crean los interfaces y de capturar los eventos que éstos generan. Una vez capturado un evento, se llama a la operación de la clase del modelo que corresponda o a otro interfaz.

Para definir el diagrama de clases de cada aplicación, se ha utilizado el mismo modelo para las tres aplicaciones, mientras que cada aplicación solo contiene las vistas que necesita y su propio controlador.

La arquitectura de la **aplicación web** contiene el paquete modelo, el paquete controladorWEB, y los paquetes que contienen las vistas web. El diagrama de clases será el siguiente:

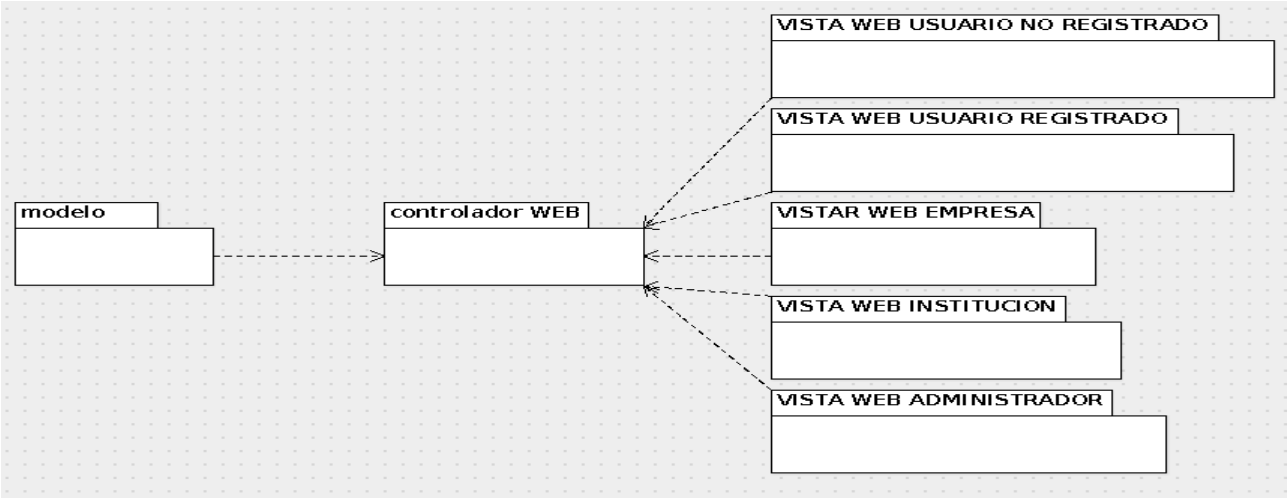


Ilustración 26: Diagrama de clases de la aplicación web

La arquitectura de la **aplicación móvil** contiene el paquete modelo, el paquete controladorMOVIL, y los paquetes que contienen las vistas móvil. El diagrama de clases será el siguiente:

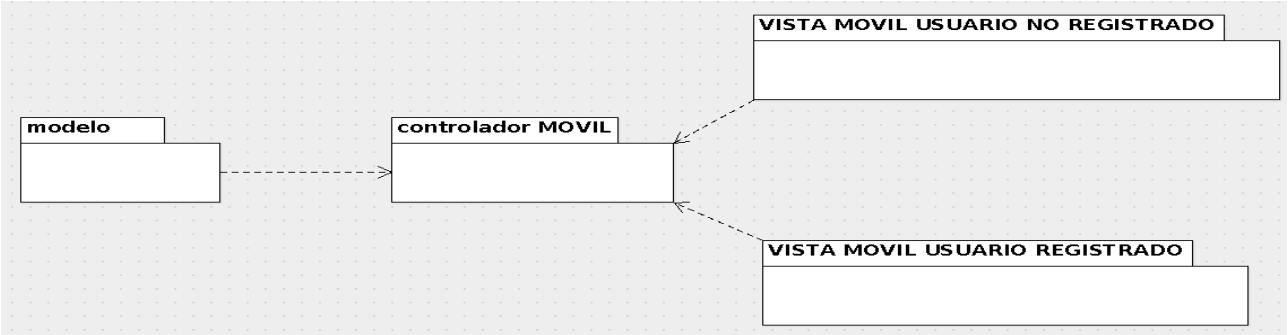


Ilustración 27: Diagrama de clases de la aplicación móvil

La arquitectura de la **aplicación widget** contiene el paquete modelo, el paquete controladorWIDGET, y los paquetes que contienen las vistas widget. El diagrama de clases será el siguiente:

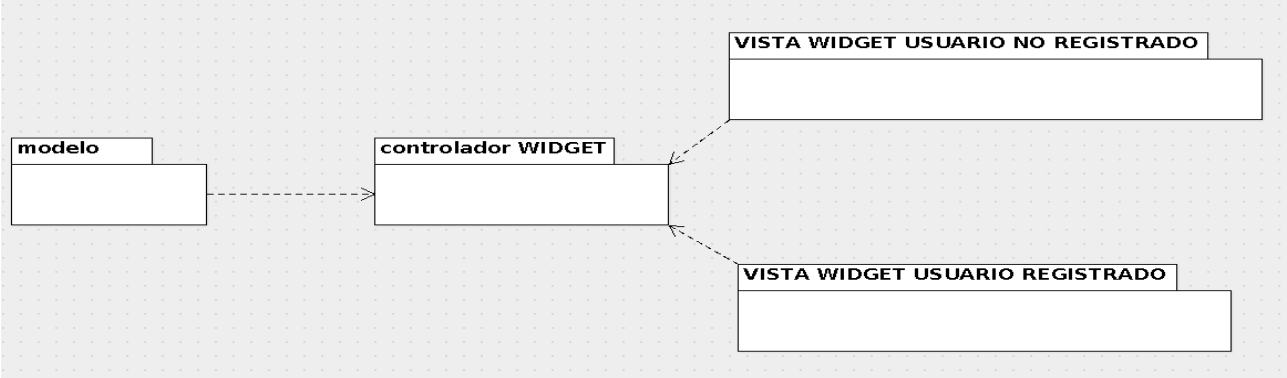


Ilustración 28: Diagrama de clases de la aplicación widget

4.2.2 Diseño de la base de datos

Dado que la información que maneja la aplicación es relativamente simple y no hay relaciones complejas entre los datos, el esquema entidad-relación puede derivarse del diagrama de clases.

Una vez hemos diseñado el diagrama de clases, para diseñar la base de datos se ha creado una tabla por cada clase que compone el modelo. Cada tabla cuenta con los mismos atributos que la clase correspondiente y como clave primaria se ha seleccionado el atributo *id*.

Para convertir una relación uno-a-muchos se ha agregado la clave primaria del lado uno a la tabla del lado muchos como clave foránea. Esa llave agregada a la tabla del lado muchos no forma parte de su llave primaria, sino que es una columna común. Por ejemplo, la tabla oferta tendrá un campo idEmpresa que corresponderá con el campo id de la empresa que publica la oferta.

El diagrama entidad-relación se presenta en la figura siguiente. Para la mejor comprensión del diagrama no se han añadido los atributos de cada entidad que se encuentran en la *ilustración 16*.

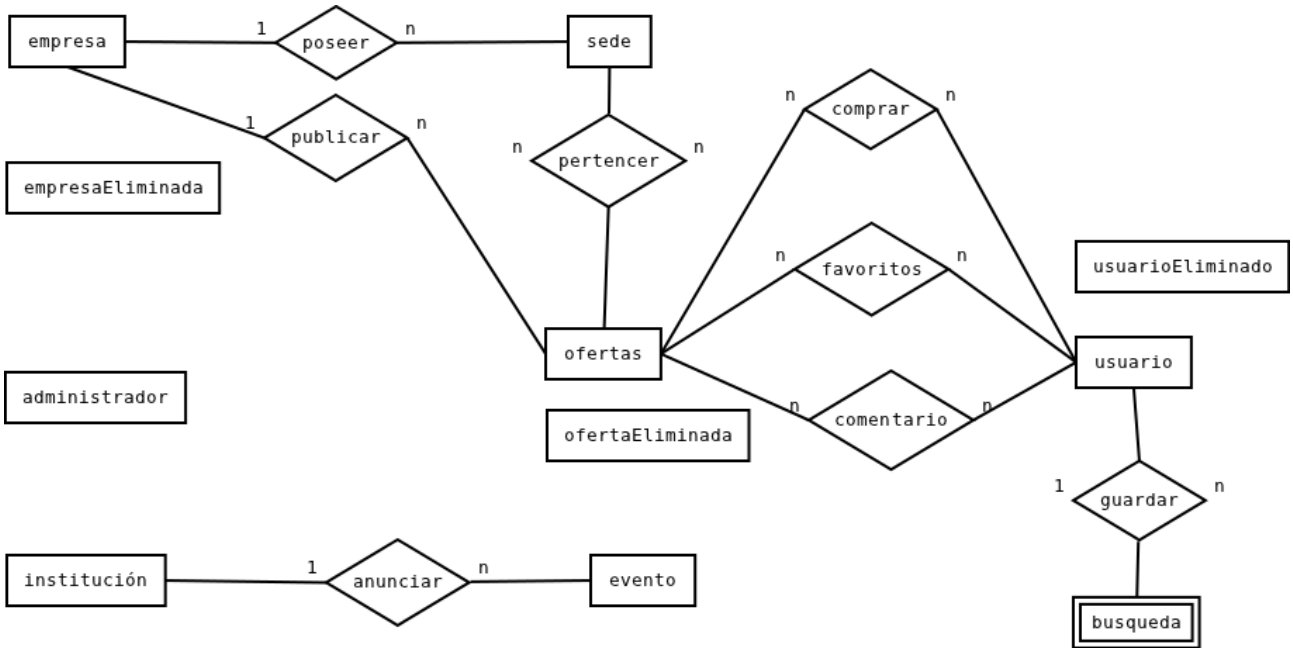


Ilustración 29: Diagrama entidad-relación

La definición de las tablas y de sus atributos se encuentran en el **anexo G**.

5 IMPLEMENTACIÓN

Durante esta etapa se han implementado, en lenguaje Java, las clases definidas durante la etapa de diseño utilizando los frameworks Vaadin y ORMLite.

En esta sección se van a comentar los aspectos de la implementación más importantes y que como desarrollador han supuesto un mayor desafío o han resultado más llamativos o novedosos. Por tanto, no se pretende que esta sección sea un tutorial de desarrollo, sino una forma de exponer los conocimientos o experiencias más importantes que se han adquirido durante el desarrollo del proyecto.

El código generado en Vaadin y ORMLite resulta difícil de entender si no se ha utilizado nunca, por eso en el **anexo G** hay varios ejemplo de programación con esta tecnología

5.1 Acceso de un usuario.

En el primer caso vamos a seguir el ejemplo que nos acompaña desde el principio y vamos a seguir el código ejecutado al identificarse un usuario en la aplicación web. Partimos de la pantalla principal en la que hay, entre otras cosas, un formulario para identificarse.

El usuario rellena el formulario con sus credenciales y hace clic en el botón "Entrar". Ésto genera un evento en la vista que se captura en el controlador, el cual llama a la función *entrada* de la clase *lógica*. Si los credenciales del usuario son correctos se accede a la aplicación principal para usuarios.

```
        //Controlador del botón "Entrar"
        view.getEntrar().addListener(new Button.clicListener() {
                /**
                 *
                 */
                private static final long serialVersionUID = 1L;

                /* Handle the clic. */
                public void buttonclic(clicEvent event) {
                        //Se llama a la funcion entrada de la clase logica.
                        int valido = logica.entrada(view,idioma);
                        if (valido == 1){//Usuario
                                //Se borran todos los elementos de la ventana y se carga la
                                 vista
                                //devuelta por la funcion aplicacionUsuario.
                                vusuario.removeAllComponents();
                                vusuario.addComponent(aplicacionUsuario(view
                                                .getNombre().getValue().toString()));
                                mainwindow.open(new ExternalResource(vusuario
                                                .getURL().toString()));
```

La función *entrada* de la clase *lógica* busca en la base de datos a que tipo de usuario pertenecen los credenciales introducidos. Devuelve el tipo de usuario si coincide con alguno y un 0 si no coincide con ninguno.

```
        /**La función entrada de la clase lógica busca en la base de datos a
         * que tipo de usuario pertenecen los credenciales introducidos.
         * @param view: vista con los credenciales del usuario
         * @param idioma: fichero que contiene los textos de la apicacion
         * @return 1 si es usuario; 2: si es empresa; 3 si es institución; 4 si es administrador
         */
        public static int entrada(EntradaView view,NProperties idioma) {
                //Se coge el dirección de correo electrónico del formulario de la vista
                String nomusuario = (String) view.getNombre().getValue()
                                .toString();
                usuario usuario = null;
                empresa empresa = null;
                institución institución = null;
                administrador administrador = null;
                //Se busca en la base de datos a que tipo de usuario pertenecen el dirección de
correo electrónico
                try {
                        //llama a la función buscarUsuario de la clase DAO
                        usuario = DAO.buscarUsuario(nomusuario);
                        //llama a la función buscarEmpresa de la clase DAO
                        empresa = DAO.buscarEmpresa(nomusuario);
                        //llama a la función buscarinstitución de la clase DAO
                        institución = DAO.buscarinstitución(nomusuario);
                        //llama a la función buscarADminsitrador de la clase DAO
```

```
                        administrador = DAO.buscarAdministrador(nomusuario);
                } catch (SQLException e) {
                        // Si ocurre algun error en el uso de la base de datos se muestra
                        //un mensaje por pantalla y se evia un dirección de correo electrónico al
adminsitrador con el fallo.
                        e.printStackTrace();
                        view.getEntrar().getWindow()
                        .showNotification(idioma.getProperty("problema"));
                        enviarError(idioma.getProperty("errorEntrar"), e);
                        return 0;
                }
                //Si borran todos los mensaje de error anteriores.
                view.getError().removeAllComponents();
                if (usuario != null) {
                        //Si el dirección de correo electrónico pertenece a un usuario
                        try {
                                //Se coge la contraseña del formulario.
                                //Si esta en blanco se muestra un mensaje de error
                                if ("".equals((String) view.getClave().getValue()
                                                .toString())) {
                                        view.añadirError(idioma.getProperty("claveIncorrecta"));
                                } else {
                                        //Si la contraseña introducida coincide con el valor de la
                                         contraseña
                                        //que hay en la base de datos para ese usuario
                                        if (usuario.getclave()
                                                        .equals((String) view.getClave().getValue()
                                                                        .toString())) {
                                                //Se actualiza la ultima conexion
                                                usuario.setUltimaconexion(usuario
                                                                .getUltimaconexion1());
                                                usuario.setUltimaconexion1(new Date());
                                                DAO.modificarUsuario(usuario);
                                                //Se devuelve un 1.
                                                return 1;
                                        } else {
                                                //Si las contraseñas no coinciden se añade un error al
                                                formulario
                                                view.añadirError(idioma
                                                                .getProperty("claveIncorrecta"));
                                        }
                                }
                                // }
                        } catch (SQLException e) {// Si ocurre algun error en el uso de la base de
                         datos se muestra
                        //un mensaje por pantalla y se evia un dirección de correo electrónico al
adminsitrador con el fallo.
                                e.printStackTrace();
                                view.getEntrar()
                                .getWindow()
                                .showNotification(
                                                idioma.getProperty("problema"));
                                enviarError(idioma.getProperty("errorEntrar"), e);
                                return 0;
                        }
```

Para buscar en la base de datos, La función *entrada* hace uso de las funciones *buscarUsuario(nombre)*, *buscarEmpresa(nombre), buscarinstitución(nombre)* y *buscarAdministrador(nombre)* de la clase *DAO*. Estas funciones buscan en la base de datos algún dato cuyo id coincida con la variable *nombre*.

```
        /**
         * busca en la tabla usuario de la base de datos el usuario cuyo id coincide con
         * id.
         * @param id: id del usuario buscado
         * @return devuelve un objeto usuario con la información. Si no existe devuelve null
         * @throws SQLException Si hay un fallo en la base de datos lanza una excepción.
         */
        public static usuario buscarUsuario(String id) throws SQLException {
                //Se inicia un objeto usuario. inicialmente con valor null
                usuario usuario = null;
                //Se pide una conexion a la base de datos
                ConnectionSource connectionSource = conectarBd();
                //se crea una instancia de la clase dao para buscar en la base de datos
                Dao<usuario, String> dao = DaoManager.createDao(connectionSource,
                                usuario.class);
                //Se busca en la base de datos el usuario
                usuario = dao.queryForId(id);
                //Se devuelve la conexion
```

```
                devolverBd(connectionSource);
                //se devuelve el usuario
                return usuario;
        }
```

```
        /**
         * busca en la tabla empresa de la base de datos la emrpesa cuyo id coincide con
         * id.
         * @param id: id de la empresa buscada
         * @return devuelve un objeto empresa con la información. Si no existe devuelve null
         * @throws SQLException Si hay un fallo en la base de datos lanza una excepción.
         */
        public static empresa buscarEmpresa(String id) throws SQLException {
                //Se inicia un objeto empresa. inicialmente con valor null
                empresa empresa = null;
                //Se pide una conexion a la base de datos
                ConnectionSource connectionSource = conectarBd();
                //se crea una instancia de la clase dao para buscar en la base de datos
                Dao<empresa, String> dao = DaoManager.createDao(connectionSource,
                                empresa.class);
                //Se busca en la base de datos la empresa
                empresa = dao.queryForId(id);
                //Se devuelve la conexion
                devolverBd(connectionSource);
                //se devuelve la empresa
                return empresa;
        }
```

```
        /**
         * busca en la tabla isntitucion de la base de datos la isntitucion cuyo id coincide
         * con id.
         * @param id: id de la isntitucion buscada
         * @return devuelve un objeto isntitucion con la información. Si no existe devuelve null
         * @throws SQLException Si hay un fallo en la base de datos lanza una excepción.
         */
        public static institución buscarinstitución(String id) throws SQLException {
                //Se inicia un objeto isntitucion. inicialmente con valor null
                institución institución = null;
                //Se pide una conexion a la base de datos
                ConnectionSource connectionSource = conectarBd();
                //se crea una instancia de la clase dao para buscar en la base de datos
                Dao<institución, String> dao = DaoManager.createDao(
                                connectionSource, institución.class);
                //Se busca en la base de datos la isntitucion
                institución = dao.queryForId(id);
                //Se devuelve la conexion
                devolverBd(connectionSource);
                //se devuelve la isntitucion
                return institución;
        }
```

```
        /**
         * busca en la tabla administrador de la base de datos el adminsitrador cuyo id coincide
         * con id.
         * @param id: id del adminsitrador buscado
         * @return devuelve un objeto adminsitrador con la información. Si no existe devuelve null
         * @throws SQLException Si hay un fallo en la base de datos lanza una excepción.
         */
        public static administrador buscarAdministrador(String id) throws SQLException {
                //Se inicia un objeto adminsitrador. inicialmente con valor null
                administrador administrador = null;
                //Se pide una conexion a la base de datos
                ConnectionSource connectionSource = conectarBd();
                //se crea una instancia de la clase dao para buscar en la base de datos
                Dao<administrador, String> dao = DaoManager.createDao(
                                connectionSource, administrador.class);
                //Se busca en la base de datos el adminsitrador
                administrador = dao.queryForId(id);
                //Se devuelve la conexion
                devolverBd(connectionSource);
                //se devuelve el adminsitrador
                return administrador;
        }
```

Una vez que la función *entrada* ha devuelto el tipo de Usuario se llama a la función *aplicacionUsuario* del controlador. Esta función llama a la vista *AplicacionEmpresaView* y se queda

esperando un evento.

```
        /**funcion que devuelve una vista con las compras de un usuario y gstiona sus eventos
         * @param id2:dirección de correo electrónico del usuario registrado
         * @return una vista
         */
        public VerticalLayout aplicacionUsuario(String id2) {
                //Se configuran las variables de sesion
                this.id = id2;
                flagTipoUsuario = 2; //Se pone el flagTpoUsuario a 2, usuario
                //Se llama a la vista
                final AplicacionEmpresaView view = new AplicacionEmpresaView(
                                ofertaDia("Registrado"), idioma);
                //Se añade el logo
                titulo(view.getTitulo());
                /**
                 * Controlador de ventos de la tab principal. Al elegir una pestaña
                 * se crea un evento que dispara esta función. Esta función
                 * intercambia la función que correspondia a la pestaña anterior por
                 * la funcion que corresponde a la pestaña elegida.
                 */
                view.getT().addListener(new SelectedTabChangeListener() {
                /**
                 * Controlador de eventos de la selección de idioma. Al seleccionar
                 * un idioma se crear un evento que dispara esta función que
                 * intercambia el fichero de idioma anterior por el del idioma
                 * elegido.
                 */
                view.getSelectIdioma().addListener(new Property.ValueChangeListener() {
                //Controlador del boton "Salir"
                view.getSalir().addListener(new Button.clicListener() {
                //se devuelve la vista
                return view.getMain();
        }
```

La clase *AplicacionEmpresaView* crea una vista perteneciente a la aplicación principal para usuarios, con una caja de pestañas con varias vistas, un selector de idiomas y un botón salir de la aplicación.

```
        Button salir; //boton "salir"
        VerticalLayout main;//vista
        VerticalLayout titulo;//espacio reservado para el logo de la aplicacion
        Select selectIdioma;//caja de seleccion de idioma
        TabSheet t;//caja de pestañas
        /**
         * Función que crea el interfaz de la pantalla principal para usuarios
         * @param ofertasDiaView: vista con las ofertas del dia
         * @param idioma
         */
        public AplicacionEmpresaView(VerticalLayout ofertasDiaView, NProperties idioma) {
                //Se crea y se configura la vista
                main = new VerticalLayout();
                main.setMargin(true);
                //Espacio reservado dentro de la vista para el logo, el boton salir
                //y la caja de seleccion de idioma
                HorizontalLayout h = new HorizontalLayout();
                h.setWidth("100%");h.setMargin(true);
                main.addComponent(h);
                //Se inicia y se añade el sspacio reservado para el logo de la aplicacion
                titulo = new VerticalLayout();
                titulo.setMargin(true);
                h.addComponent(titulo);
                //Se inicia y se añade el sspacio reservado para la caja de seleccion de idioma
                VerticalLayout EspacioIdioma = new VerticalLayout();
                h.addComponent(EspacioIdioma);
                //Se crea y se añade el boton "salir"
                salir = new Button(idioma.getProperty("salir"));
                salir.setStyleName("ofertastheme");
                h.addComponent(salir);
                //Se inicciliza la caja de selección de idioma.
                //Se añaden las opciónes a elegir, se configura y se añade al espacio
                //reservado para su visualización
                selectIdioma = new Select(idioma.getProperty("seleccionIdioma"));
                selectIdioma.addItem(idioma.getProperty("castellano"));
                selectIdioma.addItem(idioma.getProperty("gallego"));
                // select1.addItem(idioma.getProperty("administrador"));
                selectIdioma.setImmediate(true);
                selectIdioma.setNewItemsAllowed(true);
                EspacioIdioma.addComponent(selectIdioma);
                 /** Se inicializa la caja de pestañas y se añaden los contenidos.
```

```
             * Inicialmente solo "ofertasDia" tiene contenido. Los demas contenidos se añaden
             en el momento en el que pulsas sobre la pestaña. Esto se hace desde el
             controlador
             * que se encuentra en OfertasApplication en la funcion OfertasApplication.
             * Se añade a la ventana fisica principal.*/
            t = new TabSheet();
            t.setHeight("100%");
            t.setWidth("100%");
            t.addTab(ofertasDiaView, idioma.getProperty("ofertasDia"));
            t.addTab(new VerticalLayout(), idioma.getProperty("alrededor"));
            t.addTab(new VerticalLayout(), idioma.getProperty("guíaCiudad"));
            t.addTab(new VerticalLayout(), idioma.getProperty("buscar"));
            t.addTab(new VerticalLayout(), idioma.getProperty("comoFunciona"));
            t.addTab(new VerticalLayout(), idioma.getProperty("favoritos"));
            t.addTab(new VerticalLayout(), idioma.getProperty("miCuenta"));
            t.addListener(this);
            main.addComponent(t);
    }
```

5.2 Cambio de idioma

Uno de los requisitos del sistema es que debe poder utilizarse en diferentes idiomas. Para implementar este requisito se ha hecho uso de los ficheros properties de java que son ficheros de texto donde almacena por cada línea, un par clave valor que representan el nombre de la variable y su valor.

Una cadena se guarda en el fichero properties escribiéndola de la siguiente manera:

```
guíaCiudad = LA guía DE MI CIUDAD
```

Donde guiaCiudad es la clave y LA GUÍA DE MI CIUDAD es el valor. Para conseguir una cadena del fichero utilizamos el siguiente comando.

```
idioma.getProperty("guíaCiudad")
```

Se ha creado un fichero properties por cada idioma en el que se podrá mostrar la aplicación. Todos los ficheros tienen las mismas claves y se diferencian en los valores, ya que cada uno contiene los valores en su idioma.

Guardando las cadenas en un fichero properties tenemos varias ventajas. Una es que, como ya hemos visto, se puede hacer una aplicación multi-idioma fácilmente. Otra es que si una cadena aparece varias veces en la aplicación, y la queremos modificar, basta con cambiarla en el fichero properties.

Cuando el usuario cambia el valor del campo de selección de idioma de la pantalla principal se crea un evento que se captura en la función *aplicacionUsuario* del controlador. Una vez capturado, se llama a la función *cambiarIdioma* de la clase *lógica* para cargar el fichero properties correspondiente al idioma seleccionado y se hace un reset de la pantalla principal.

```
    /**
     * Controlador de eventos de la selección de idioma. Al seleccionar
     * un idioma se crear un evento que dispara esta función que
     * intercambia el fichero de idioma anterior por el del idioma
     * elegido.
     */
    view.getSelectIdioma().addListener(new Property.ValueChangeListener() {
            /**
             *
             */
            private static final long serialVersionUID = 1L;

            public void valueChange(
                            com.vaadin.data.Property.ValueChangeEvent event) {
                    if (event.getProperty().getValue() != null) {
                            //Se llama a la clase cambiar idioma de la clase logica
                            idioma = logica.cambiarIdioma(view.getSelectIdioma()
                            .getValue().toString(),idioma);
                            /**
                            Se hace un reset de la pantalla. Primero se borran todos los
                            elementos.
                             */
                            view.getSelectIdioma().getWindow().removeAllComponents();
```

```
                        /**
                         * Despues se vuelve a cargar ya con el nuevo idioma.
                         */
                        vusuario.addComponent(aplicacionUsuario(id));
                    }
                }
            });
```

La función *cambiarIdioma* de la clase *lógica* llama a la clase NProperties con el idioma seleccionado. Esta función devuelve una instancia de un fichero properties con los textos de la aplicación en el idioma seleccionado.

```
    /**Funcion para cambiar el idioma de la aplicacion
     * @param seleccion: idioma seleccionado
     * @param idioma: instancia de fichero properties con los textos del idioma anterior.
     * @return: instancia de fichero properties con los textos del nuevo idioma.
     */
    public static NProperties cambiarIdioma(String seleccion,
                    NProperties idioma) {
            /**
             * Si la opción elegida es "castellano" se carga el fichero de idioma castellano
             */
            if (seleccion
                            .equals(idioma.getProperty("castellano"))) {
                    /**
                     * Nproperties es una clase que carga el fichero de idioma que le pides.
                     * En este caso el español.
                     */
                    idioma = new NProperties("ES");
            }
            /**
             * Si la opción elegida es "gallego" se carga el fichero de idioma gallego
             */
            if (seleccion
                            .equals(idioma.getProperty("gallego"))) {
                    idioma = new NProperties("GL");
            }
            return idioma;
    }
```

La clase NProperties es una clase que devuelve una instancia de fichero properties con los textos en idioma seleccionado.

```
    /**
     * Constructor de la clase NProperties       *
     * @param idioma: idioma seleccionado
     */
    public NProperties(String idioma) {
            if (idioma.equals("ES")) {
                    // devuelve una instancia de dichero
                    // properties con los textos en español
                    getProperties("../Properties/español.properties");
            } else if (idioma.equals("GL")) {
                    // devuelve una instancia de dichero
                    // properties con los textos en gallego
                    getProperties("../Properties/gallego.properties");
            } else {
                    // por defecto devuelve una instancia de dichero
                    // properties con los textos en gallego
                    getProperties("../Properties/español.properties");
            }
    }

    /* se leen las propiedades */
    public void getProperties(String idioma) {
            try {
                    this.load(getClass().getResourceAsStream(idioma));
            } catch (IOException ex) {
            }
    }
```

5.3 Geolocalización

Cuando el usuario se conecta desde un móvil a la aplicación móvil, ésta recoge sus coordenadas y calcula cual es la provincia en la que esta usando una funcionalidad de google maps.

Cuando se inicia una aplicación desarrollada con Vaadin en un navegador se obtienen los

detalles del navegador (que navegador es, tamaño, etc) mediante la función onBrowserDetailsReady. Uno de estos detalles son las coordenadas.

```
/**
 * Funcion que contiene los detalles del navegador en el que se ejecuta la aplicacion
 * en la que esta.
 */
@Override
public void onBrowserDetailsReady() {
       //obtiene localizacion
       mainwindow.detectCurrentPosition(new PositionCallback() {
           public void onSuccess(Position position) {
               //Coordenada X
                double latitude  = position.getLatitude();
               //Coordenada /
                double longitude = position.getLongitude();
               //Precision
                double accuracy  = position.getAccuracy();
```

A partir de ellos podemos saber cual es la provincia desde la que se esta usando la aplicación. Para eso usamos la funcionalidad Geocode de Google Maps pasándole como parámetros las coordenadas y nos devolverá contenido con información de nuestra posición, calle, ciudad, provincia, etc.

```
//Preparamos una petición a la funcionalidad Geocode de Google Maps
URI uri = new URI("http://maps.google.com/maps/api/geocode/json?
latlng="+latitude+","+longitude+"&sensor=true");
                        httpGet.setURI(uri);
//ejecutamos la peticion
 HttpResponse httpResponse = httpClient.execute(httpGet);
 //cogemos el contenido de la respuesta y lo metemos en un buffer
 InputStream contenido = httpResponse.getEntity().getContent();
 BufferedReader bufferedReader = new BufferedReader(new
  InputStreamReader(contenido));
 //Pasamos el contenido del buffer a la cedena texto
 String ligneLue;
  ligneLue = bufferedReader.readLine();
 String texto="";
 while(ligneLue !=null)
         {
             texto=texto+ligneLue;
             texto=texto+"\n";
             ligneLue = bufferedReader.readLine();
         }
```

Pasamos la información del contenido a tipo *String*.

```
//Pasamos el contenido del buffer a la cedena texto
 String ligneLue;
  ligneLue = bufferedReader.readLine();
 String texto="";
 while(ligneLue !=null)
         {
             texto=texto+ligneLue;
             texto=texto+"\n";
             ligneLue = bufferedReader.readLine();
         }
```

Dentro de la cadena *texto* hay mucha información sobre nuestra posición. Lo que nos interesa en este caso es la provincia en la que se encuentra el usuario. Esta información se encuentra tras la subcadena *long_name* de la siguiente manera:

```
"long_name" : "Corunna"
```

Por lo tanto, para conseguir la localización, hay que buscar el valor de la clave *long_name* en la cadena *texto* y guardarlo en la variable global *Provincia*.

```
//separamos el texto por espacios en blanco
 String[] arrayBuscar = texto.split(" ");
//Se busca la clave "long_name" que es la que contiene la Provincia
       for (int i = 0; i < arrayBuscar.length; i++) {
       if(((arrayBuscar[i]).indexOf("long_name"))!=-1){
              //cuando se ha encontrado se guarda su valor en la variable global Provincia
              Provincia = arrayBuscar[i+2];
                     //adaptamos el valor devuelto al castellano.
                     if ((Provincia.indexOf("Corunna"))!=-1){Provincia="La Coruña";}
              }
```

La variable global *Provincia* será utilizada para seleccionarla por defecto en las distintas búsquedas que se pueden realizar desde la aplicación móvil.

6 PRUEBAS DEL SISTEMA

Aunque el Documento de Casos de Uso muestre detalladamente la forma en la que debe funcionar el sistema, siempre se escapan algunos detalles que se deben corregir en una etapa de pruebas exhaustivas. En esta sección realizaremos las pruebas funcionales, las pruebas de volumen y las pruebas de rendimiento diseñadas para garantizar un correcto funcionamiento del sistema.

Se han definido las pruebas funcionales a realizar sobre las distintas secuencias de los casos de uso definidos. Es decir, si un caso de uso tiene 1 secuencia normal y 2 secuencias alternativas se definirán 3 casos de prueba para ese caso de uso.

Para definir un caso de prueba se usa la siguiente plantilla.

Caso de prueba: CP-<Número Secuencia> Nombre	
Descripción	< *Objetivo y alcance de la prueba.* >
Precondiciones	< *Estado del sistema antes de comenzar la prueba.* >
Entradas	< *Lista de estímulos y datos específicos a introducir.* >
Elementos relacionados	< *Nombre caso de uso que ejercita.* >
PROCEDIMIENTO DE PRUEBA	
Actor	**Sistema**
< *Describir paso a paso las instrucciones para ejecutar el caso de prueba.*>	< *Respuesta esperada del sistema*>
RESULTADO OBTENIDO	
< *Se describe el resultado obtenido, adjuntando pantallazos si es necesario*>	

Tabla 4: Plantilla para definir un caso de prueba

Para las pruebas de rendimiento y las pruebas de volumen se han insertado 100000 entradas en cada tabla de la base de datos para comprobar que los tiempo de respuesta están dentro de un intervalo aceptable y que el sistema trabaja perfectamente con grandes volúmenes de datos.

Una vez definidos todos los casos de prueba e insertado los datos en la base de datos, se ejecutan los casos de prueba para comprobar que el sistema cumple con los requisitos iniciales.

Un caso de prueba puede tener dos resultados: positivo o negativo. Si el resultado de todos los casos de prueba relacionados con un caso de uso han sido positivos, se dan por cumplidos los requisitos relacionados con el caso de uso. Si el resultado ha sido negativo, se analiza y corrige el error, y se vuelve a realizar el mismo test. Esta operación se ha repetido hasta que todos los casos de prueba han dado positivo.

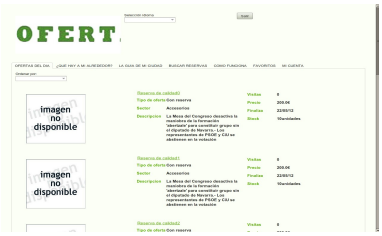
Una vez que todos los casos de prueba han obtenido resultado positivo, se puede decir que se ha cumplido con el objetivo y que el sistema funciona correctamente.

Siguiendo con el ejemplo de la evolución del requisito vamos a ver los casos de prueba relacionados con el *CU-2 identificarse.*

Caso de prueba: CP-1 identificación de un usuario con datos correctos	
Descripción	Se realiza el acceso al sistema de un usuario introduciendo datos incorrectos.
Precondiciones	El usuario dispone de conexión a internet. El sistema funciona correctamente.

	El usuario se encuentra en el sistema como usuario no logueado. El usuario está registrado con dirección de correo electrónico fuertes5@gmail.com y clave 0perand0. El usuario se encuentra en la página de acceso.
Entradas	E-mail:fuertes5@gmail.com Clave: 0perand0
Elementos relacionados	*CU-2 Acceder*

PROCEDIMIENTO DE PRUEBA	
Actor	**Sistema**
1. introduce los credenciales del campo entrada. 2. Pulsa el botón “entrar”.	El usuario ha accedido al sistema. Verifica que existe una cuenta de tipo de usuario usuario con ese dirección de correo electrónico y esa contraseña y muestra la pantalla principal para usuarios. El sistema no sufre incidencias y su funcionamiento es adecuado.

RESULTADO
Positivo 

Caso de prueba: CP-1 identificación de un usuario sin cuenta	
Descripción	Se realiza el acceso al sistema de un usuario introduciendo unos datos que no están en la base de datos.
Precondiciones	El usuario dispone de conexión a internet. El sistema funciona correctamente. El usuario se encuentra en el sistema como usuario no logueado. El usuario está no registrado. El usuario se encuentra en la página de acceso.
Entradas	E-mail:fuerte5@gmail.com Clave: 0perand0
Elementos relacionados	CU-2 Acceder

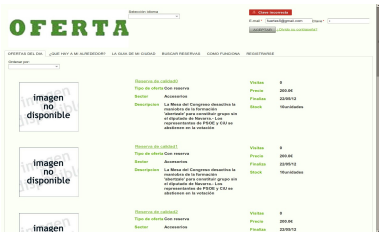
PROCEDIMIENTO DE PRUEBA	
Actor	**Sistema**
1. introduce los credenciales del campo entrada.	Muestra un mensaje de error informando de que no existe la

2. Pulsa el botón “Entrar”	cuenta.

RESULTADO OBTENIDO
positivo 

Caso de prueba: CP-1 identificación de un usuario con contraseña incorrecta	
Descripción	Se realiza el acceso al sistema de un usuario introduciendo datos incorrectos.
Precondiciones	El usuario dispone de conexión a internet. El sistema funciona correctamente. El usuario se encuentra en el sistema como usuario no logueado. El usuario está registrado con contraseña “0perand0”. El usuario se encuentra en la página de acceso.
Entradas	E-mail:fuertes5@gmail.com Clave: a
Elementos relacionados	CU-2 Acceder

PROCEDIMIENTO DE PRUEBA	
Actor	**Sistema**
1. introduce los credenciales del campo entrada. 2. Pulsa el botón “Entrar”	Muestra un mensaje de error informando que la contraseña es errónea

RESULTADO OBTENIDO
positivo 

Caso de prueba: CP-1 identificación de un usuario con fallo en la base de datos.	
Descripción	Se realiza el acceso al sistema de un usuario cuando la base de datos esta caída.
Precondiciones	El usuario dispone de conexión a internet. La base de datos esta caída. El usuario se encuentra en el sistema como usuario no logueado. El usuario está registrado. El usuario se encuentra en la página de acceso.
Entradas	E-mail:fuertes5@gmail.com Clave: 0perand0
Elementos relacionados	CU-2 Acceder

PROCEDIMIENTO DE PRUEBA	
Actor	**Sistema**
1. introduce los credenciales del campo entrada. 2. Pulsa el botón “Entrar”	Falla al insertar en la base de datos. Muestra un mensaje informando de dicho error.

RESULTADO OBTENIDO	
positivo	

Tabla 5: ejemplo de caso de prueba

Los casos de prueba de acceso de una empresa, una institución y un administrador también están relacionados con el CU-2 Acceder. Éstos casos y todos los demás casos de prueba definidos y ejecutados se detallan en el **anexo E**.

7 CONCLUSIONES Y LINEAS FUTURAS

7.1 Conclusiones

Cuando se inició este proyecto se plantearon una serie de objetivos genéricos ligados al prototipo que se quería desarrollar. Sin embargo, el propio proceso de desarrollo también tenía sus propios objetivos, siendo el primer objetivo fundamental el de conocer la plataforma de Vaadin, puesto que inicialmente no tenía conocimiento alguno de la misma.

La búsqueda de una aplicación que fuera suficientemente útil para ser usada, abordable por una persona en un plazo de tiempo razonable y variada como para aprender los aspectos típicos de esta plataforma no fue fácil, pero fue un reto que me permitió poner en práctica la mayoría de conocimientos aprendidos durante mis estudios y mejorar mi capacidad de auto- aprendizaje.

El objetivo principal del proyecto se ha visto satisfecho ampliamente pues se ha conseguido

cumplir todos los requisitos al desarrollar un sistema de ofertas en el que empresas e instituciones naciones pueden ofrecer sus productos, servicios y eventos para que los clientes puedan informarse o consumirlos a través de internet o de un teléfono móvil. Ademas, se ha implementado un apartado genérico, en el que se permite la publicación y venta de productos y servicios y la visualización de eventos de las empresas e instituciones que pertenezcan al entorno de un Ayuntamiento específico.

Esta idea resultó ser el núcleo de todo este proceso de desarrollo que se ha explicado a lo largo del documento, abarcando desde el análisis de los requisitos de la aplicación hasta la implementación de la misma.

Es evidente que, fruto de todo este trabajo, se ha conseguido satisfacer también los objetivos planteados de conocer los detalles necesarios de la plataforma Vaadin y establecer los pasos necesarios para desarrollar aplicaciones para ella.

Desde un punto de vista personal, la realización de este proyecto me ha servido para descubrir un gran interés por el mundo del desarrollo en el campo de las aplicaciones web. Si tenemos en cuenta la situación actual de este mercado, donde la mayor parte de las empresas se dedican al desarrollo de aplicaciones web, resulta que tengo ante mí una muy buena oportunidad de futuro.

En conclusión, la realización de este proyecto ha servido para satisfacer todos los objetivos planteados al inicio del mismo, a la vez que ha abierto una puerta hacia un futuro profesional que, al comienzo del mismo, no existía.

7.2 Lineas futuras

Tras la finalización del proyecto han surgido algunas líneas de trabajo bastante interesantes y que sería oportuno evaluar su realización.

- Crear una aplicación nativa en Android, iOs y Windows Phone a partir de la aplicación móvil desarrollada.
- Ofertas con desplazamiento. Se ejecuta la compra que lleva asociada una empresa de transporte que entrega en el domicilio los productos comprados. Habría que establecer algún tipo de acuerdo con una empresa de transportes.
- Posibilidad de reservas por tramos de tiempo:
  - reserva por horas: para peluquerías, campos de fútbol, piscinas,etc.
  - reserva por días: para hoteles, campings, etc.

8 BIBLIOGRAFÍA

Fuentes electrónicas:

[1] https://vaadin.com/ (Consulta: 26/09/2011)
[2] http://www.postgresql.org.es/ (Consulta: 26/09/2011)
[3] http://ormlite.com/ (Consulta: 26/09/2011)
[4] http://www.avajava.com/tutorials/lessons/how-do-i-use-a-jdbc-realm-with-tomcat-and-mysql.html?page=2 (Consulta: 21/10/2011)
[5] http://www.infor.uva.es/~jvegas/cursos/bd/sqlplus/sqlplus.html (Consulta: 21/10/2011)
[6] http://www.pgcon.org/2011/schedule/attachments/194_pgcon2011-pgdroid.pdf (Consulta: 21/10/2011)
[7] http://www.postgresql.org/docs/8.1/static/sql-createtable.html (Consulta: 21/10/2011)
[8] http://www.guía-ubuntu.org/index.php?title=PostgreSQL (Consulta: 21/10/2011)
[9] http://www.siadsoft.com/index.php?option=com_content&view=article&id=45:como-instalar-postgresql-en-ubuntu&catid=1:latest-news (Consulta: 21/10/2011)
[10] http://www.postgresql.org/docs/9.1/static/sql-createindex.html (Consulta:

21/10/2011)
[11] http://www.monografias.com/trabajos-pdf2/sistema-gestion-base-datos-postgresql/sistema-gestion-base-datos-postgresql.pdf (Consulta: 21/10/2011)
[12] https://developers.google.com/maps/documentation/javascript/v2/services?hl=es-ES#ReverseGeocoding (Consulta: 21/10/2011)
[13] https://developers.google.com/maps/documentation/geocoding/?hl=es#GeocodingRequests (Consulta: 21/10/2011)
[14] http://hanzcocchi.net/proceso-desarrollo-software-orientado-objetos/ (Consulta: 6/10/2011)
[15] http://www.oocities.org/es/kagutierbcv/ads/t1/contenido.htm (Consulta: 6/10/2011)
[16] http://www.cannes.itam.mx/Alfredo/Espaniol/Publicaciones/MINT/Requisitos.pdf (Consulta: 6/10/2011)
[17] http://fermat.usach.cl/~msanchez/comprimido/OBJETOS.pdf (Consulta: 6/10/2011)
[18] http://unadocenade.com/una-docena-de-webs-para-comprar-con-descuento/ (Consulta: 26/09/2011)
[19] http://danielpecos.com/docs/mysql_postgres/x108.html (Consulta: 21/10/2011)
[20] http://www.groupon.es/ (Consulta: 26/09/2011)
[21] http://ofertix.com/escaparate (Consulta: 26/09/2011)
[22] http://es.privalia.com/public (Consulta: 26/09/2011)
[23] http://es.buyvip.com/index.jsp (Consulta: 26/09/2011)
[24] http://es.letsbonus.com/ (Consulta: 26/09/2011)
[25] http://advertise.planeo.com/es/ (Consulta: 26/09/2011)